A Multi-Purpose App for Hospital Induction

Haow Jern Tee [1]

MSc Computer Science

Dean Mohemadally

Submission date: 04/09/2019

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Hospital inductions can be confusing for junior doctors. This project aims to build an app to improve the experience through creating a maps system to help with navigation, and to use Augmented Reality to make it a more entertaining experience. Besides that, the project also involves creating easy to use pipelines for Great Ormond Street Hospital staff to upload assets as needed.

A mobile app was built to improve the induction experience for junior doctors at Great Ormond Street Hospital. It assists in navigation through providing a user-interface to access user-made hospital maps and to search for locations within the hospital. It uses Augmented Reality to place and look for user-made 3D assets. A website was built as a user-interface with CRUD functionality for maps and to view existing data, whereas 3D assets were set up to be uploaded as Assetbundles through the Unity application. Data was stored in a MySQL database, whereas 2D and 3D assets were stored using Azure Blob Storage. To facilitate communication between user applications and storage, a REST-like server was built.

Acknowledgements

I would like to express my gratitude to my supervisors, Dr. Dean Mohemadally, Gemma Molyneux and Daiana Bassi for their invaluable guidance and advice throughout the project, as well as to Justin Poisson and his team of the Post Graduate Medical Education department at Great Ormond Street Hospital for feedback on the app design, and for enabling this project to be possible. Thank you all.

Acronyms

API Application Programming Interface. 8, 10, 26, 41

AR Augmented Reality. 2, 5, 8–10, 12, 18–20, 22, 43, 45–47

ASA Azure Spatial Anchors. 5, 9, 10, 26, 36, 40, 45

GOSH Great Ormond Street Hospital. 5, 10, 12, 18, 44–46

PGME Post Graduate Medical Education. 5, 18, 19, 44

REST Respresentational State Transfer. 5, 10, 26, 28

SDK Software Development Kit. 9

VR Virtual Reality. 8

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The project attempts to improve on existing methods for hospital induction for junior doctors. Great Ormond Street Hospital (GOSH) spans multiple buildings and may be disorientating to a newcomer, hence an effective induction is paramount to ensure that they are as familiar with the hospital's layout as quick as possible. Induction is held every month and organised by the Post Graduate Medical Education (PGME) department and they would like to improve this experience using Augmented Reality, a better maps system for navigation, to integrate their physical treasure hunt system into the app and to have an easy way for the current hospital staff to upload their own assets to be used by the app, and

A Unity mobile app using Azure Spatial Anchors (ASA) for AR was built. Navigation was improved through the ability to view multiple maps of floors and buildings in the app. A pipeline to upload and load 3D assets was created using Unity's Assetbundles. A web app written in Angular was built for the uploading of maps. The database used was MySQL, hosted using Azure Database For MySQL, and the storage to store maps and Assetbundles was Azure Blob Storage. A REST-like API server, written in Express.js was also built to facilitate communication between the mobile and web app.

Initial project plans such a treasure hunt system, an achievement system, a user login system and a way-finding system were not done due to time constraints. However there will be some relevant discussion in a brief section under Design and Implementation, chapter 4 to discuss some design choices.

1.1 Aims and Goals

The aims were to:

- Learn Unity development and programming in C#.
- Learn how to develop for Android Apps.

- Learn how to connect server for both websites and apps to communicate to a database.
- Learn how to develop in Angular for Website.
- Learn how to develop Augmented Reality using Unity.

The goals were to:

- Develop a website connecting to a backend.
- Develop an Android application connecting to the same backend.
- Develop a pipeline to dynamically load resources.
- Develop an Android application using augmented reality.

1.2 Approach

Software planning and design were carried out using the Unified Process methodology [1]. This involves a series of defined development steps - gathering of requirements, analysis and design, implementation, testing and deployment. This project was well suited to this methodology compared to Agile methods like Scrum [2] because the requirements for the project were unlikely to change during implementation, and it allowed stakeholders to know which stage the project is at. To visualise my progress in between development cycles, I used Trello [3] along with Kanban [4] method to organise my project.

1.3 Report Overview

The rest of the report is divided into the following sections:

2. **Background and Research**:

   In this chapter, the necessary background information to understand the report will be given. The author will also discuss relevant research done to support the choice of technologies used.

3. **Requirements Gathering and Analysis**:

   In this chapter, an extensive documentation of necessary project requirements through use of MosCow, use cases and etc. will be given.

4. **Design and Implementation**:

   In this chapter, the high-level architecture, general designs and noteworthy implementations will be discussed.

5. **Testing**:

   In this chapter, testing techniques such as Unit Testing and noteworthy test cases will be discussed.

6. **Conclusion and Project Evaluation**:

   A summary of the project outcome, general thoughts and future recommendations.

Chapter 2

Background & Research

2.1 What is Augmented Reality?

AR is the technology that describes the integration of 3D virtual objects with a 3D real environment in real time [5]. It has many real-world applications, ranging from military [6] to manufacturing [7] to games such as Pokemon Go [8]. AR is usually implemented with head-mounted displays such as the Oculus Rift and mobile devices. What makes a mobile device suited for AR is that it already has the necessary hardware to support it, such as a camera, a GPS, processors and other sensors. These factors are standard in today's mobile devices, forming a very inexpensive method for consumers to use AR compared to purchasing AR dedicated hardware.

AR should not be confused with Virtual Reality (VR), as both are different concepts that are encompassed by the term 'Mixed Reality' [9]. For the purposes of understanding, the 'difference' between VR and AR is that, in VR, a user interacts with virtual elements in a virtual world, whereas in AR, a user interacts with virtual elements in the real world.

2.2 Similar Apps

Based on research, there were two AR Treasure Hunt apps, UniRallye [10] and Hunt [11] implemented before and published as research papers. UniRallye was a prototype developed for University orientation using Vuforia, a marker-based AR technique. The authors reported that users of the app enjoyed the experience and found it useful. Hunt was another prototype built thier authors' university library orientation, using the Metaio SDK with marker-based AR. They reported that participants were more motivated and engaged during the orientation. For navigation, Google Maps would be the most similar, however as GPS cannot be used in buildings, its Application Programming Interface (API) cannot be used.

2.3 Augmented Reality Choice

There are two main types of AR, categorised based on implementation:

- Marker AR
- Marker-less AR

Markers refer to fiducial markers similar to barcodes or uploaded images that are supplied to the device for detection and tracking. They tend to be relatively easy to implement and maintain. A popular library is Vuforia [12]. Marker-less AR refer to methods that do not use supplied images, these can be location-based AR or object based AR, i.e. a user places an object in an environment, and the device records information about its environment to display the object as soon as the same environment is pointed at. This is known as creating an anchor. In marker-based AR, for a multi-user experience, where multiple devices can view and interact with the same object placed by another device, users will have to create an anchor and upload it to the cloud, known as a cloud anchor. As cloud anchors are necessary for multi-user experiences, the only Software Development Kit (SDK)s that support this are ARCore [13], ARKit [14], ARFoundation [15] and Azure Spatial Anchors (ASA) [16]. The downsides to using cloud anchors are that cloud anchors requires a wireless connection to work but it is not an issue as there is a consistent wireless connection throughout the hospital.

Marker-based AR, even though it is simpler to implement, was not desirable as there may be locations in the hospital where markers cannot be placed. The free version of Vuforia also has a mandatory watermark. ARKit was ruled out as there was no support for hybrid development including iOS and Android devices. ARCore was great, but cloud anchors did not persist for longer than 24 hours, which was not suitable for this project. ARFoundation is built on top of ARCore and ARKit, but that also meant the same issues as ARCore. On further research, ASA allowed cloud anchors to persist indefinitely, and with the most recent release, is built with the support of ARFoundation, which allows the possibility of hybrid development. It works by allowing mobile devices to create cloud anchors in the user's native device using ARCore or ARKit, and converting it to a ASA cloud anchor. It does the conversion in the background. Besides that, ASA also has the advantage of allowing way-finding without the explicit use of a GPS, which can provide a benefit to navigation.

2.4 Website Development

The choice of framework used is Angular, combined with HTML, Javascript and CSS. This was chosen to learn about its component system and to use its dependency injection framework, which helps in unit testing. Angular Material Design was also used to incorporate standard styling. Visual Studio Code was used as the editor with its plethora of extensions.

2.5 Mobile Development

The choice of language is Unity using C#. Unity was chosen as it has a stronger support for AR development over Unreal Engine. ASA also supports only Unity or native development, and choosing Unity allows development for both platforms. However, this project will only focus on the development for Android.

2.6 Backend

The database chosen is MySQL, hosting on the cloud using Azure Database for MySQL. To store assets such as images and 3D models required for the app, Azure Blob Storage will be used. It was chosen as it is very scalable for unstructured data (blobs) and storage needs can be flexible depending on needs [17]. This scalability benefit was true in general for other Azure services as well. Besides that, both GOSH and UCL use Azure Services in their technologies, hence it should be easier to integrate. To create and run SQL scripts, MySQLWorkBench was used; To check and test blob storage, Microsoft Azure Storage Explorer was used. Scripts were created and tested using MySQLWorkBench.

A relational database was chosen over other non-relational databases, such as MongoDB (a document-based database) because there are many relationships that need to be recorded. Even though MongoDB can scale better than MySQL, as it makes use of distributed processing (horizontal scaling) better [21], it is not necessary as the load required to reach this difference is not expected for the life cycle of this app.

To communicate between the web app, mobile app, database and storage, a REST-like API was created using Express.JS as the server-side framework written in Javascript.

This choice of technologies is similar to a MEAN stack [22], which stands for MongoDB, ExpressJS, Angular and Node.js, except that MongoDB is replaced with MySQL. The advantage of this stack is that the same language is used throughout, i.e. Javascript (TypeScript is very similar to Javascript), while using the same Node.js runtime enviroment.

2.7 Version Control

Github is used for code files, and Dropbox is used for other files such as images and documents.

2.8 Mockup

Figma was used, due to its familiarity and its powerful tools in creating mock ups.

2.9 Dynamic Loading of Assets

To use 3D Assets in an Unity app, the relevant 3D models created with different computer graphics software must be compiled and loaded into Unity. Loading them into a Unity scene creates a GameObject that is readable by Unity and this is needed to load the asset. To load assets at run time, Unity requires the use of Assetbundles, which is a type of file that contains assets. Hence, assets must be compiled, built into Assetbundles, saved to the server, and then retrieved in the app to load it during run time. A script can be created to automate the building and saving to the server. However, an issue is that it still requires the use of the Unity application, but the client has someone who is dedicated to creating graphical assets, and the process of using Unity to upload 3D assets is simple. The advantage is that they have the flexibility of choosing the software they want to use to create the 3D asset as long as Unity has the support for it.

Chapter 3

Requirements Gathering and Analysis

3.1 Problem Statement

The main aim of this project is to improve the hospital induction experience for junior doctors using AR, as well as providing a method to help with navigation and an easy way to dynamically load user-created assets into the app.

3.2 Requirements

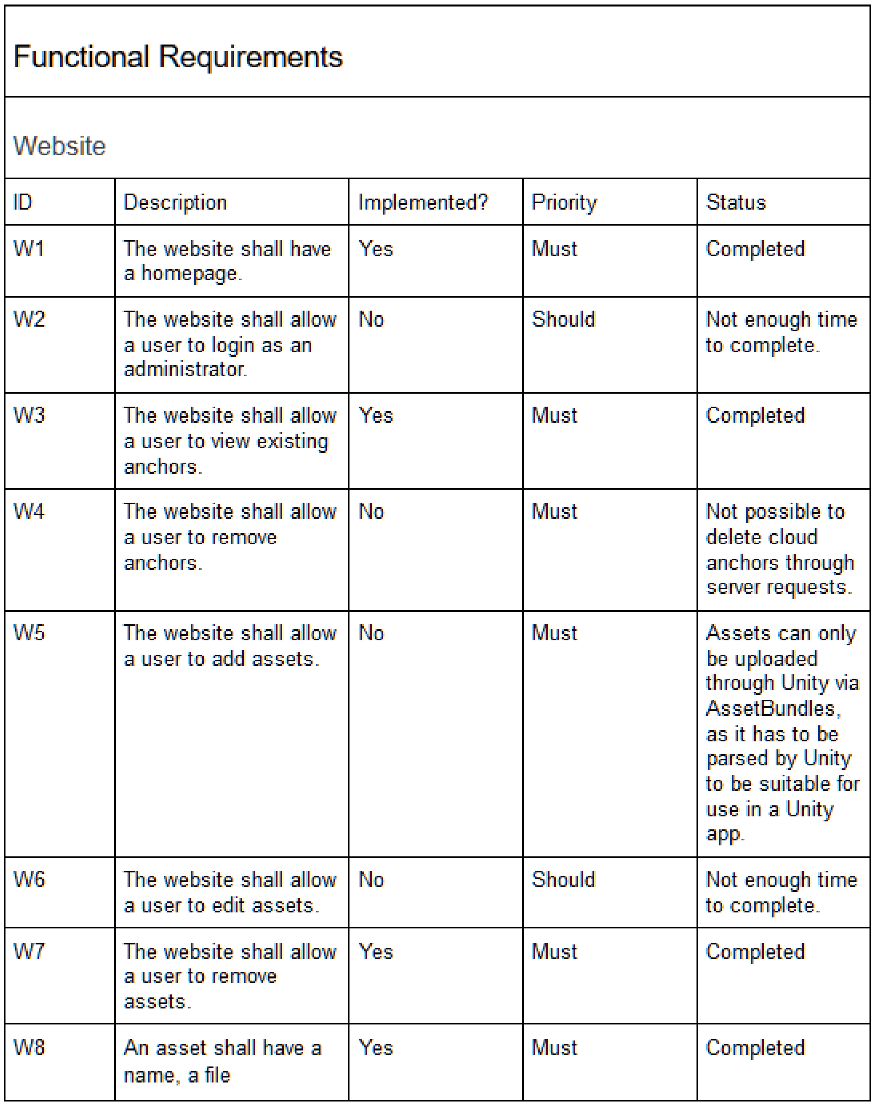
Through interviews with GOSH supervisors and the PGME committee, a full set of requirements was gathered. They are categorised into functional and non-functional requirements, and given a priority level based on the MosCoW technique, an example is shown on Figure 3.1, the rest is shown in Appendix C.1.

3.3 Use Cases

Based on the requirements, use cases were created, shown on Table 3.2. To visualise the interactions between users and system, use case diagrams were generated on Figures 3.1 and 3.2. Detailed use cases were created for the main functionality of the app. An example of a detailed use case for the 'View Maps' use case is shown on Figure 3.3. For the full list of detailed use cases, please refer to Appendix C.2.

3.4 General Design Solution

Based on the requirements and the use cases, the following is the general design of the five systems used by the app. The Maps System and AR System have been implemented, but

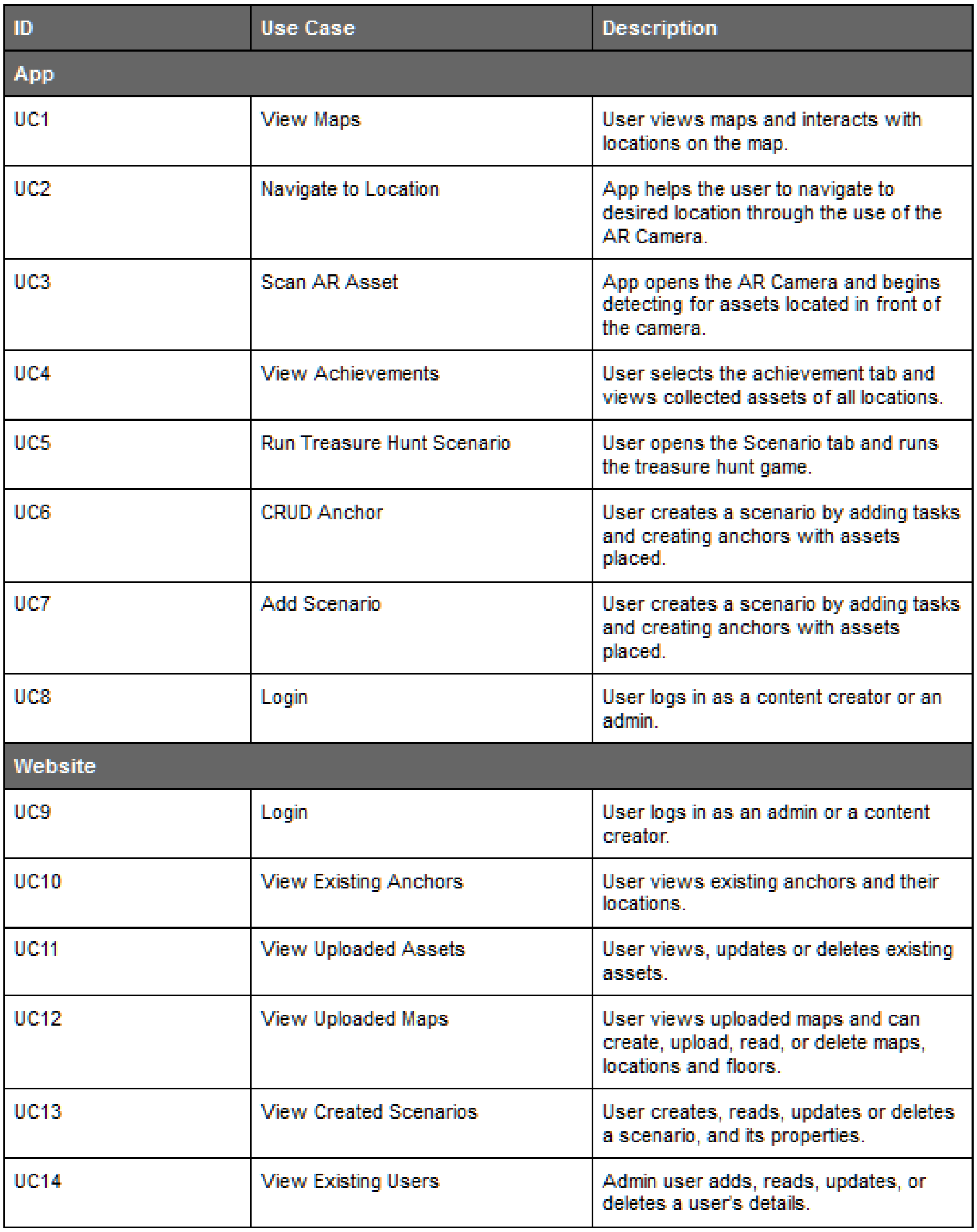


Functional Requirements

Website

ID	Description	Implemented?	Priority	Status
W1	The website shall have a homepage.	Yes	Must	Completed
W2	The website shall allow a user to login as an administrator.	No	Should	Not enough time to complete.
W3	The website shall allow a user to view existing anchors.	Yes	Must	Completed
W4	The website shall allow a user to remove anchors.	No	Must	Not possible to delete cloud anchors through server requests.
W5	The website shall allow a user to add assets.	No	Must	Assets can only be uploaded through Unity via AssetBundles, as it has to be parsed by Unity to be suitable for use in a Unity app.
W6	The website shall allow a user to edit assets.	No	Should	Not enough time to complete.
W7	The website shall allow a user to remove assets.	Yes	Must	Completed
W8	An asset shall have a name, a file	Yes	Must	Completed

Table 3.1: An excerpt of the full requirements. The rest is shown in Appendex C.1



ID	Use Case	Description
App		
UC1	View Maps	User views maps and interacts with locations on the map.
UC2	Navigate to Location	App helps the user to navigate to desired location through the use of the AR Camera.
UC3	Scan AR Asset	App opens the AR Camera and begins detecting for assets located in front of the camera.
UC4	View Achievements	User selects the achievement tab and views collected assets of all locations.
UC5	Run Treasure Hunt Scenario	User opens the Scenario tab and runs the treasure hunt game.
UC6	CRUD Anchor	User creates a scenario by adding tasks and creating anchors with assets placed.
UC7	Add Scenario	User creates a scenario by adding tasks and creating anchors with assets placed.
UC8	Login	User logs in as a content creator or an admin.
Website		
UC9	Login	User logs in as an admin or a content creator.
UC10	View Existing Anchors	User views existing anchors and their locations.
UC11	View Uploaded Assets	User views, updates or deletes existing assets.
UC12	View Uploaded Maps	User views uploaded maps and can create, upload, read, or delete maps, locations and floors.
UC13	View Created Scenarios	User creates, reads, updates or deletes a scenario, and its properties.
UC14	View Existing Users	Admin user adds, reads, updates, or deletes a user's details.

Table 3.2: Use Cases.

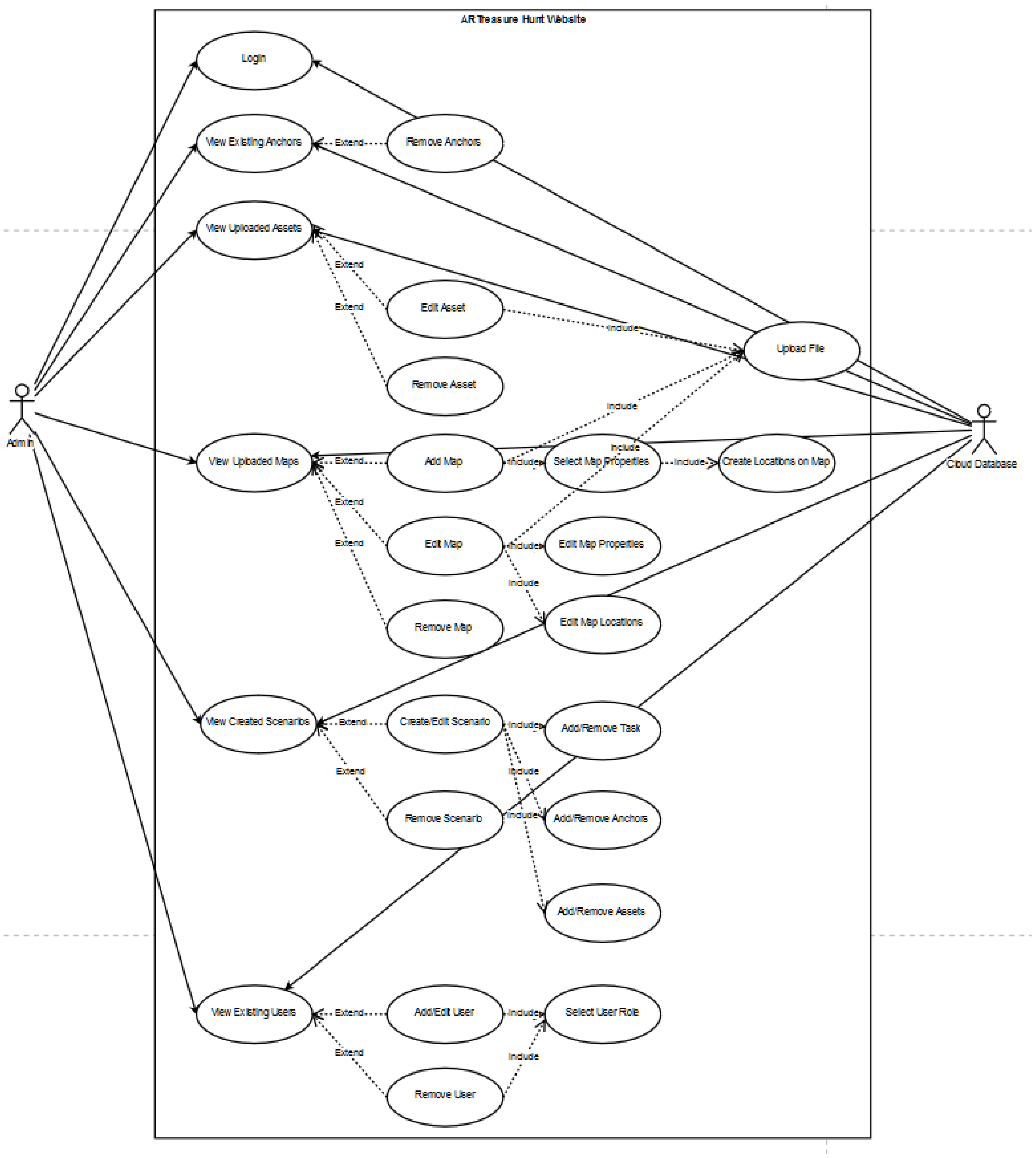


Figure 3.1: Use Case Diagram - Web App.

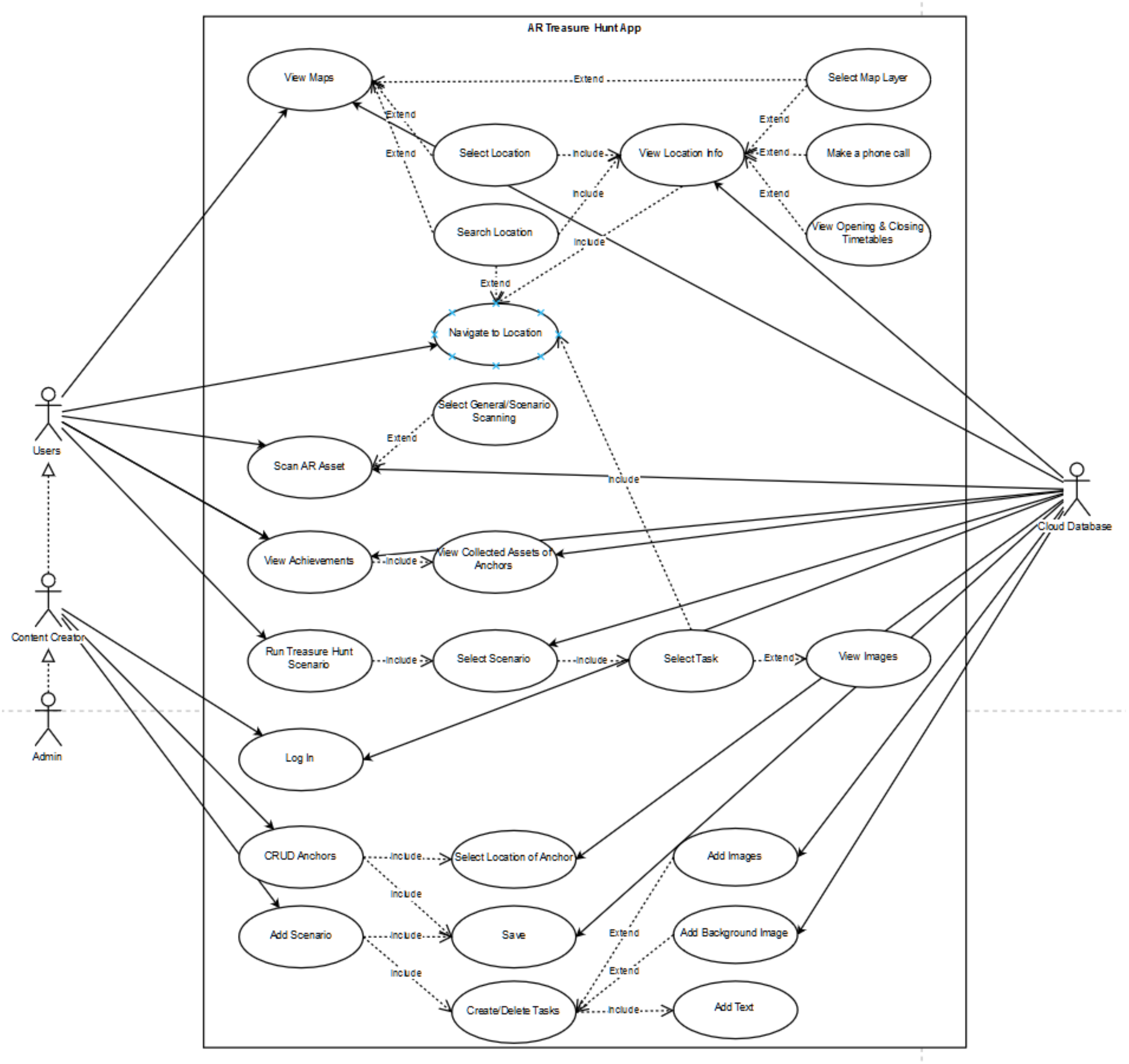


Figure 3.2: Use Case Diagram - Mobile App.

Use Case	View Maps
ID	UC1
Brief Description	User views maps and interacts with locations on the map.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	1. User selects map tab. 2. User views the default map. 3. User selects a location 4. App displays location info **If** building was selected and the location is in the building: 5. User selects the enter button. 6. User selects the building floor. 7. App displays the map view of the selected building's floor and pings the location.
Alternative Flows	1. User selects a location name. 2. App pings the location on the map view. 3. User selects location. 4. App displays location info **If** building was selected and the location is in the building: 5. User selects the enter button. 6. User selects the building floor. 7. App displays the map view of the selected building's floor and pings the location.
Pre-Conditions	None
Post-Conditions	None

Table 3.3: Detailed Use Case Diagram - View Maps. A detailed description of a use case outlined in Table 3.2.

the rest of the app has not, however a discussion of this is important as it determined how the app will look and be structured, and as well as influence the website, server and database design. Some implementation considerations have been made, and this will be the discussed in Unimplemented Work under Section 4.8.

1. Maps System

   The goal of the maps system is to allow a user to view different locations on a map, and use that information to help them navigate. It should allow users to view information such as opening and closing times of the locations. The maps system is in charge of loading maps from the server, and creating views for the maps. The user will be able to see the default map, which should be the bird-eye view of GOSH. A user can click on a location indicated on the map to view its information, and to enter the location if it is a building. Once inside the building, the user can view multiple maps of different floors, and view the locations in each floor. The user is able to search for different locations through a drop down list.

2. AR System

   This opens the camera and allow the user to scan for created cloud anchors. It allows the admin user to create cloud anchors, assign assets, delete cloud anchors, and upload them to the server.

3. Scenario System

   This is the treasure hunt design. The app should allow the user to select a few scenarios to run. Each scenario consists of multiple tasks, and each task has a clue about what is needed to be found. Every time the user finishes a task, the next one opens up.

   The choice of the scenario and current task determines what cloud anchors will be looked for by the AR System and how to navigate between locations by the Navigation System.

   The decision to separate this system from the AR system was due to an interview with PGME. During induction, junior doctors will have a series of lectures throughout the day, and may not have the dedicated time to follow a treasure hunt game. Hence this is separated so that there is something users can use in the event that a scenario is not suitable to be ran.

4. Achievement System

   This should help users to be more invested, and is one of the ways of gamification. The idea is that when an asset is found, the user 'collects' the asset. They are able to view collected and uncollected assets.

5. Navigation System

   When a user has a location selected and selects this, the app will direct the user to the location. An alternative flow is when the user is running a scenario, and selects this, the app will navigate the user to the required location of the Task.

3.5 Mock-Ups & Iterations

Based on the general designs of the app, some storyboards and illustrations were created to get feedback from supervisors and PGME. Once storyboards were well received, mock ups were then created.

A few examples of mocks are shown in Figures 3.5, 3.3 and 3.4. These went through a few iterations, for example, Figures 3.3 and 3.4 show the initial mock-ups of the map view and add anchors AR Camera view. The final app design is shown in Figures 3.6 and 3.7. For map view, a 'drop up' button was changed to a 'drop down' as it was more intuitive. For add anchors AR Camera view, more buttons were needed such as the delete button. The add assets section changed from a shift left and right selection list to a drop down list, the choice made was that a drop down list is easier to navigate through when there are a larger number of assets. This also helps to improve performance as in the original case, assets would have to be loaded (or other methods used) to display the assets. However in the current version, an asset would only need to be loaded when an asset name is selected. Besides that, the UI for this screen has been reduced, as AR tracking performs better with a bigger camera view.

To view and interact with the mocks, please refer to the references [18] and [19] to access the mock ups on Figma.

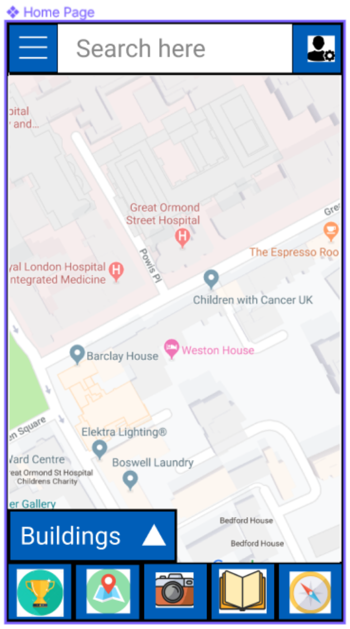


Figure 3.3: Mock Up: Mobile App Map Screen. The hamburger menu is a drop down to for users to select a location. A relevant map will be displayed for the selected location. The drop up 'Buildings' represent the map layer currently viewed. A user can switch between different map layers to view, e.g. different floors of a building. Designed in Figma [19].

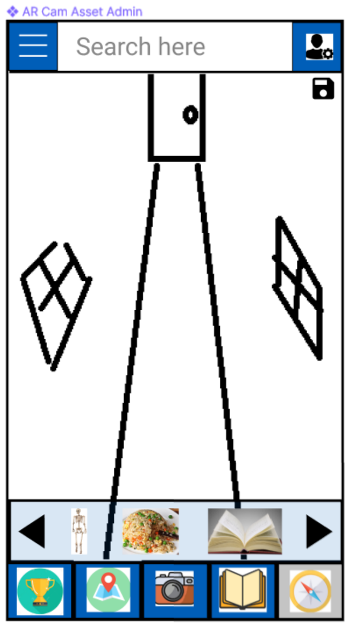
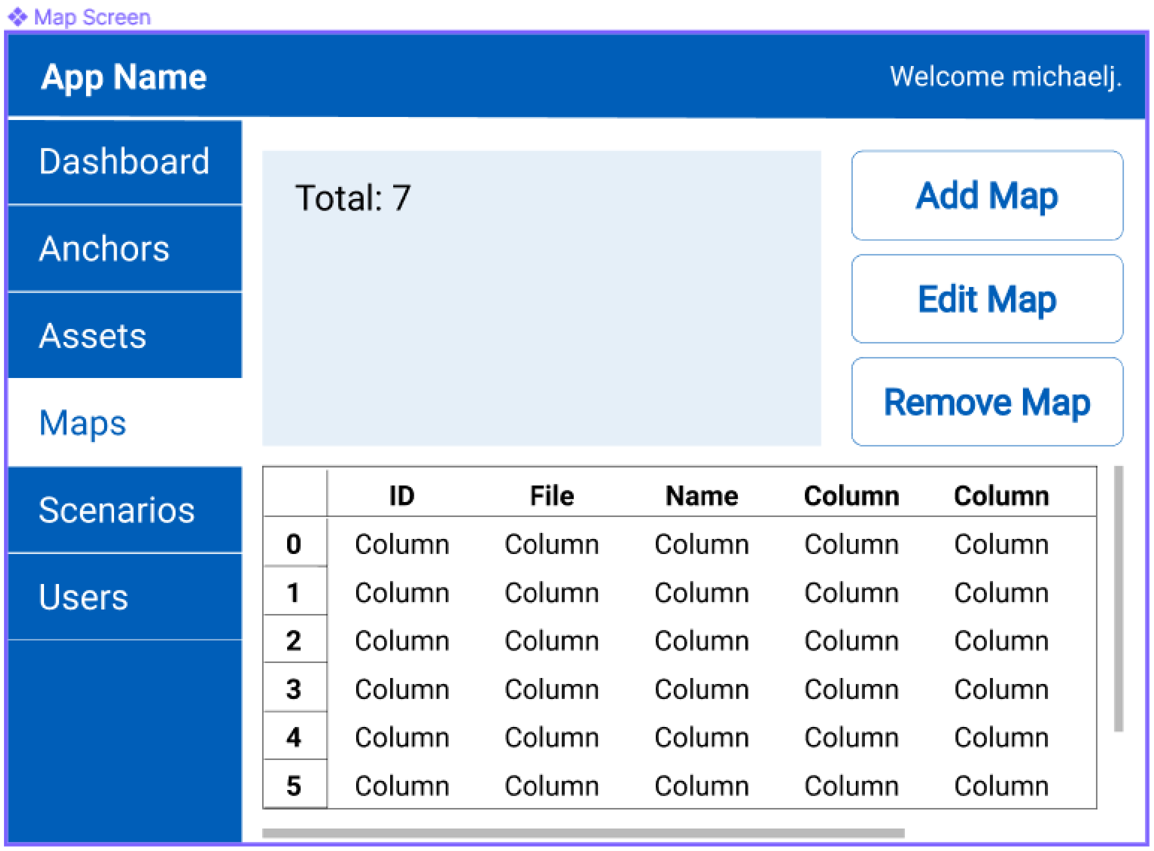


Figure 3.4: Mock Up: Mobile App Add AR Assets Screen. Allows a user to drag an asset and place it on the camera screen (denoted by the paint diagram), to create an AR Anchor. Designed in Figma [19].



	ID	File	Name	Column	Column
0	Column	Column	Column	Column	Column
1	Column	Column	Column	Column	Column
2	Column	Column	Column	Column	Column
3	Column	Column	Column	Column	Column
4	Column	Column	Column	Column	Column
5	Column	Column	Column	Column	Column

Figure 3.5: Mock Up: Web App Maps Screen. It displays the tables gathered from the database. The area with 'Total' is an empty area designed for future analytics use, and the three buttons redirect the user to different pages to perform their respective functions. Designed in Figma [18].

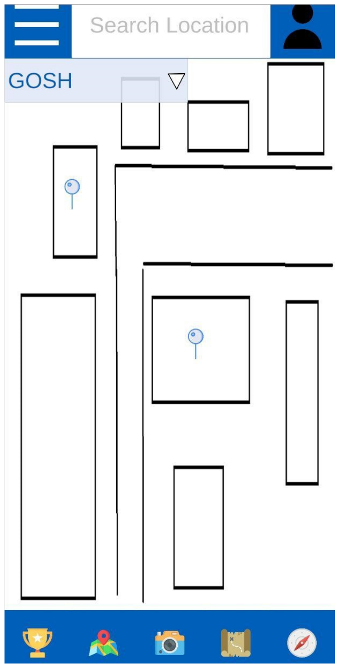


Figure 3.6: Implementation: Maps View main screen. The functionality is similar to the mock up mentioned in Figure 3.3, but the drop up has changed to a drop down for a better UI design. The small icons on the map represent clickable locations.

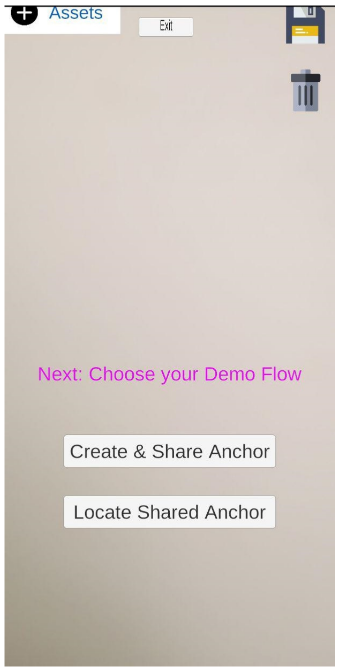


Figure 3.7: Implementation: Add AR Anchors Screen. Much of the UI is removed as performance AR tracking is better with a bigger camera view. A 'delete' anchor button and an 'add asset' button is used.

Chapter 4

Design and Implementation

4.1 Overview

The following sections will be a discussion of the general design solution, which gives the context of how the app should function based on the requirements outlined previously, the High-Level Architecture, the design of the implementation of the project, and a discussion about interesting aspects of the Back-end, Server, Web App and Mobile App designs.

4.2 High-Level Architecture

The architecture of the project was a three tier architecture [20] forming the client-side, server-side, and backend, shown in Figure 4.1. Client-side consists of the mobile app (Unity, C#) and web app (Angular, Javascript, HTML, CSS); Server-side consists of the web-server (Express.JS, Javascript), and the back-end consists of Azure Blob Storage, Azure Database For MySQL and Azure Spatial Anchors. The client website, and the web server were both hosted in the cloud using Azure Web App Services, along with Azure Blob Storage, Azure Database For MySQL and Azure Spatial Anchors.

4.3 Database & Storage

Database design was done and the final iteration of the Entity Relationship Diagram in third normalisation form is shown in Figure 4.2. An example change was that, the 'assets' table used to not have a 'blob_name' column, but during implementation of Azure Blob Storage, there needed to be a reference to the URL where the asset is stored, which it now represents.

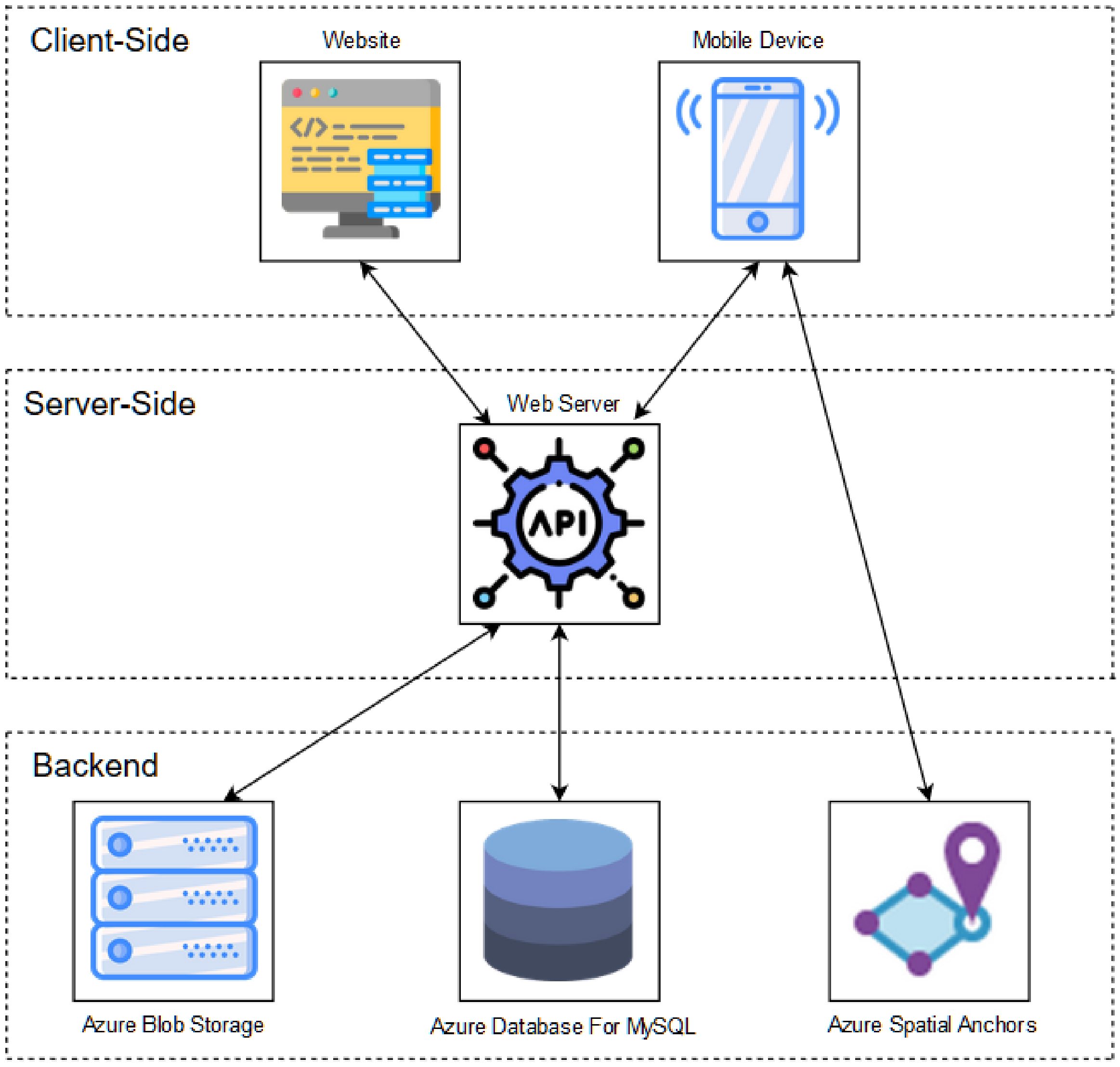


Figure 4.1: High Level Architecture of the project. Arrows represent data transfer.

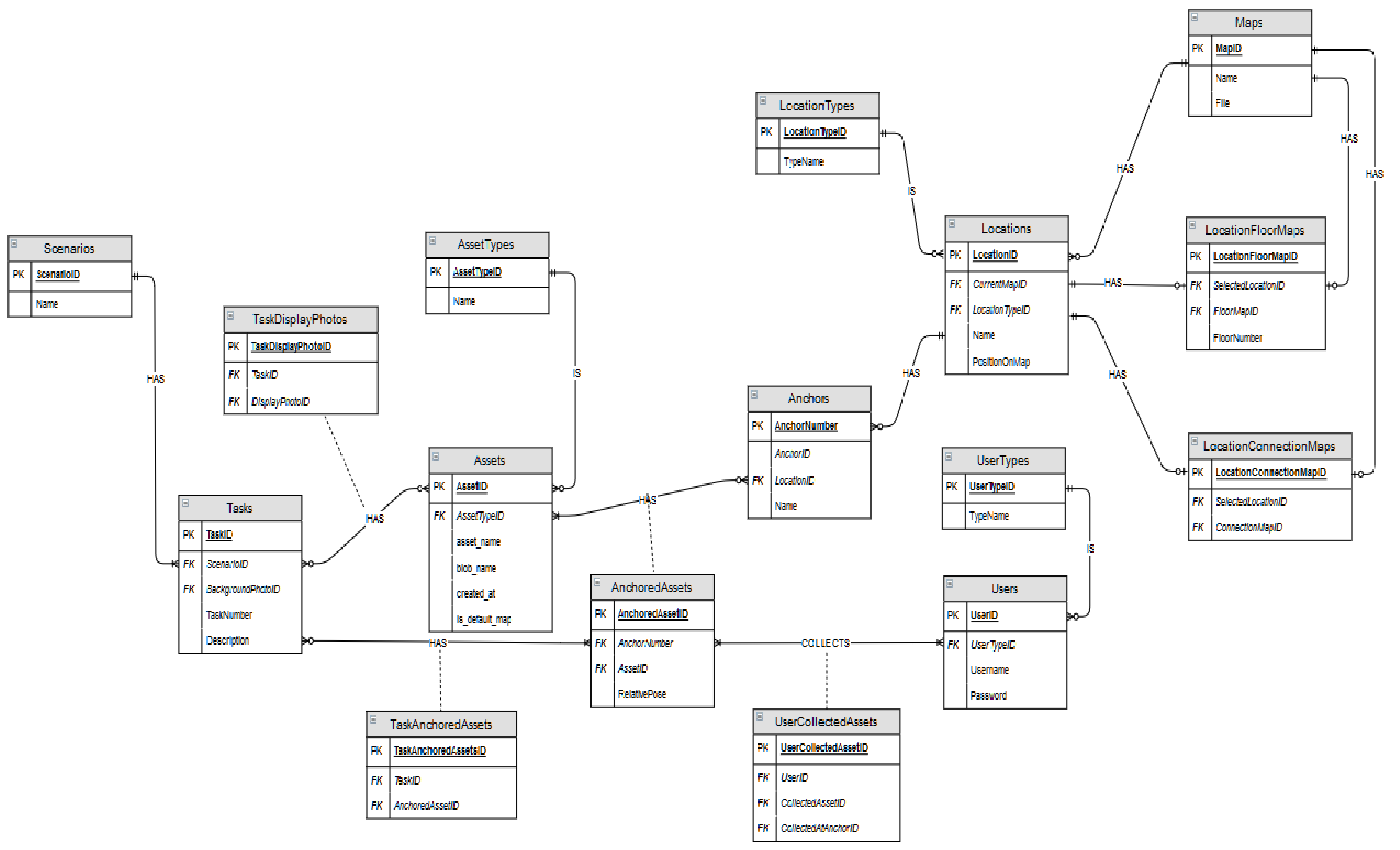


Figure 4.2: Entity Relationship Diagram developed for database design.

An 'initialisation.sql' script was created which contained the default values needed to be stored into the 'Type' tables, and deletes and resets the database, allowing for a quick and convenient way of making iterations on the database. Appropriate field properties such as when to set 'NOT NULL' and when to set 'CASCADE — SET NULL' for 'ON DELETE — ON UPDATE' were considered. An excerpt is shown on Figure 4.3. When an 'asset_typeID' row is deleted, the referenced value in the assets table should be set null, as an admin may want to change existing types. Otherwise, the row will also be deleted and data lost.

4.4 Azure Spatial Anchors

All of the conversion from native anchors to cloud anchors and the vision processing algorithm for environment tracking and detection were done in the cloud using ASA service.

4.5 Web Server

An API provides a uniform interface to make request calls through a server to operate on data. The standard HTTP Request Methods are GET, POST, PUT and DELETE.

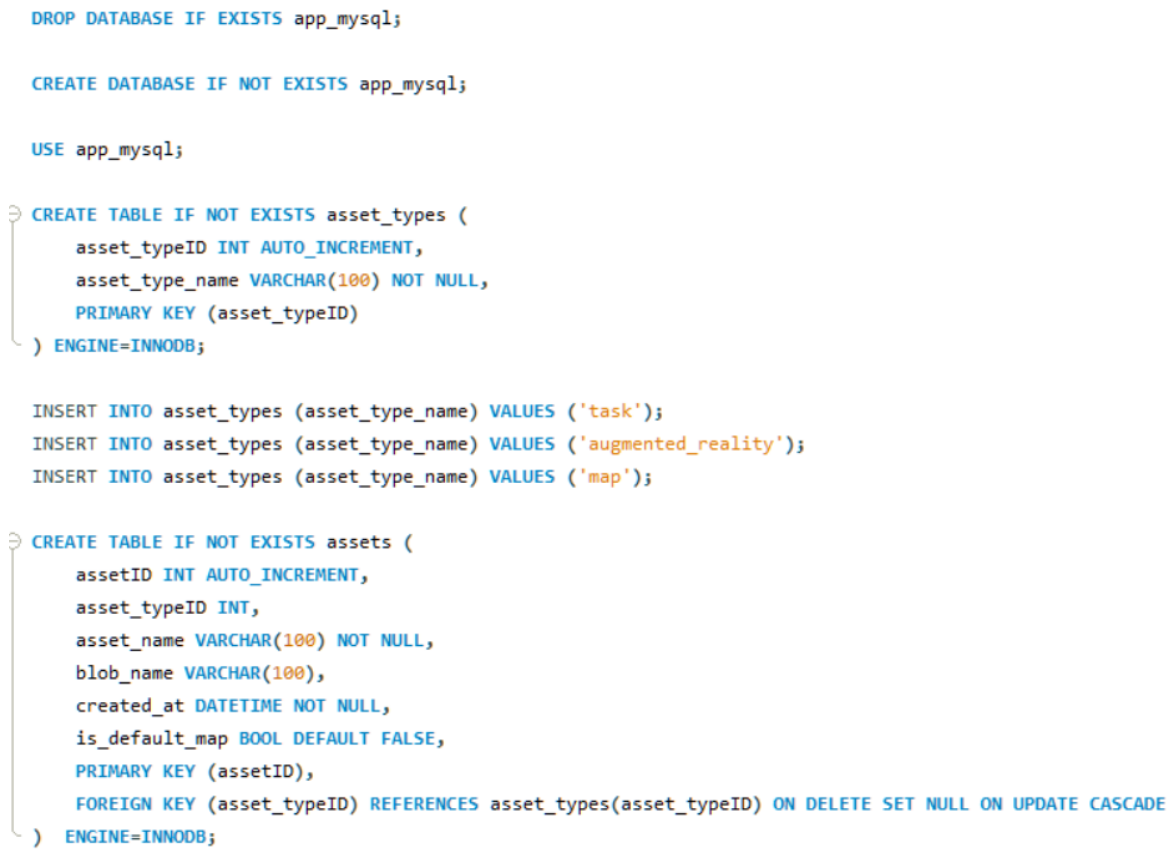
As there were multiple client-server communications, a web server was needed to be created to facilitate this. It is written in Express.JS as it supports Respresentational State Transfer (REST) services, and multiple calls. The server was written with the REST architecture in mind. It is a "set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations" [23]. For an API to be considered REST-ful, it should follow these guidelines:

1. Client-Server

   A REST application has a client-server architecture. This enables separation of concerns, improving portability. The server in this project consists of files - server.js, config.js, database.js, blob_access.js and other files named by the table they are performing operations on. Server.js represents the client in that it holds the URLs for client applications to access and retrieve data, but the method of retrieval of data from the database and storage is determined by the other files. These files require the use of config.js, database.js, and blob_acess to access data. A detailed high-level architecture is shown in Figure 4.4.

2. Stateless

   All information required to service a client's request is contained in the request, hence there is no session data about the client stored in the server. In this app, to perform specific requests to the database, client applications must supply the correct request



```sql
DROP DATABASE IF EXISTS app_mysql;

CREATE DATABASE IF NOT EXISTS app_mysql;

USE app_mysql;

CREATE TABLE IF NOT EXISTS asset_types (
    asset_typeID INT AUTO_INCREMENT,
    asset_type_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (asset_typeID)
) ENGINE=INNODB;

INSERT INTO asset_types (asset_type_name) VALUES ('task');
INSERT INTO asset_types (asset_type_name) VALUES ('augmented_reality');
INSERT INTO asset_types (asset_type_name) VALUES ('map');

CREATE TABLE IF NOT EXISTS assets (
    assetID INT AUTO_INCREMENT,
    asset_typeID INT,
    asset_name VARCHAR(100) NOT NULL,
    blob_name VARCHAR(100),
    created_at DATETIME NOT NULL,
    is_default_map BOOL DEFAULT FALSE,
    PRIMARY KEY (assetID),
    FOREIGN KEY (asset_typeID) REFERENCES asset_types(asset_typeID) ON DELETE SET NULL ON UPDATE CASCADE
)  ENGINE=INNODB;
```

Figure 4.3: 'Initialisation.sql'. A SQL script to create database, define tables and add in default values. A similar script was used to create the test database for testing.

with the appropriate route, parameters and data to 'server.js'. An example is shown in Figure 4.7 where 'mapId' is a required parameter in the route for the GET request.

3. Uniform-Interface

   Having a uniform interface simplifies and decouples the system. This is represented by the defined routes in the project. Each route is a unique route to access a particular resource and to perform an operation. A route is defined by a URL - e.g `http://www.server.com/api/maps-management/maps`. When a GET request is made to this route, it will retrieve all 'map' objects from the back-end. In this project, HTTP request methods are defined to allow the usage of the same routes with different purposes:

   - GET: Retrieve resources.
   - POST: Upload a resource.
   - PUT: Update a resource.
   - DELETE: Delete a resource.

   A parameterised route makes it easy to request for specific resources, e.g. for `http://www.server.com/api/maps-management/maps/1`, the request will operate on a map with id = 1. A full list of routes and their respective functionality is described in Appendix C.4.

4. Layered System

   Each component in the system cannot see beyond the layer that they are interacting with. In this project, 'server.js' does not know about 'blob_access.js' and vice versa. The connections to the databases and storage is not known by the client.

5. Cacheable

   Responses sent back by a REST application can be cached by the client. This allows successive requests to use the same cached response, improving network efficiency. However this program does not explicitly allow cached data, as the requirements for this would be to add appropriate headers in the response, such as the time that the requested resource was created, the time it was last updated, and etc. [24] Hence this server is REST-like.

4.5.1 Security

Basic security features were implemented. CORS (cross-origin-resource sharing) was enabled to only receive requests from specified URL addresses - `http://localhost:4200` and

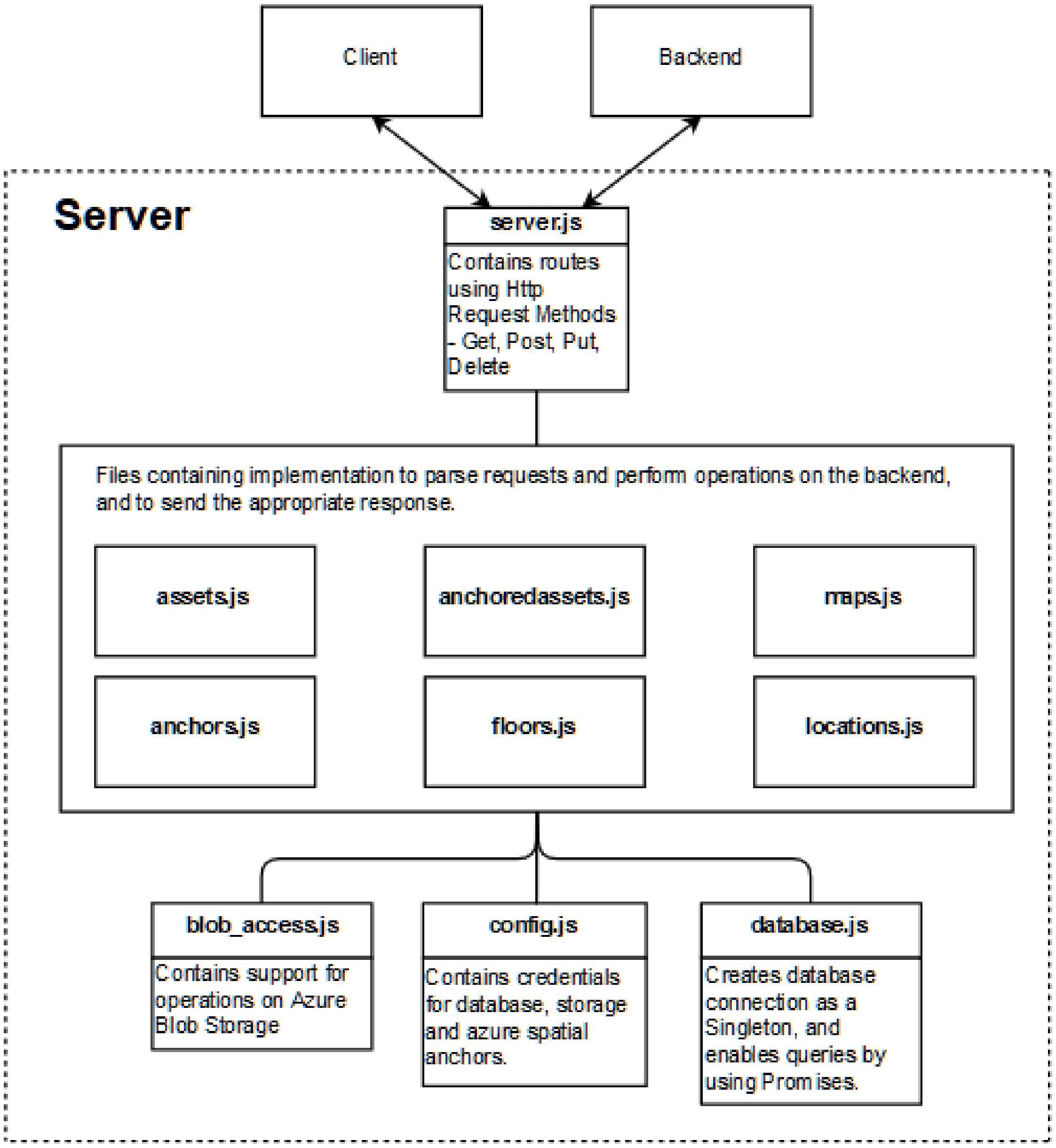

Client
Backend
Server
server.js
Contains routes using Http Request Methods - Get, Post, Put, Delete
Files containing implementation to parse requests and perform operations on the backend, and to send the appropriate response.
assets.js
anchoredassets.js
maps.js
anchors.js
floors.js
locations.js
blob_access.js
Contains support for operations on Azure Blob Storage
config.js
Contains credentials for database, storage and azure spatial anchors.
database.js
Creates database connection as a Singleton, and enables queries by using Promises.

Figure 4.4: Server file-system architecture. Each 'implementation' file, e.g. assets.js, anchoredassets.js and etc. export their functions to be used in server.js by the appropriate route.

`https://app-treasure-hunt-website.azurewebsites.net'`. The latter is used for local development and the latter is during production or for mobile testing.

From Figure 4.7, the use of '?' in the SQL string escapes the input (in this case it is mapId), this helps to prevent SQL injections.

Some safety features were implemented too, such as that for a map, it is uploaded to blob storage is first before a database record is created. If there is an issue with creating a database record, the uploaded blob will be deleted.

4.5.2 Database

A singleton class can be loosely defined as class with one and only one single instance throughout the program. The database.js (Figure 4.5) contains the Database class which is implemented as a singleton so that queries will use the same instance of the class, and the benefit is that it does not matter when this class is instantiated first, it will still return the same instance. A database pool of connections is created, and a connection is provided upon a request and released back to the pool when a request is sent. The default query function, existing from the MySQL library, is modified to be able to use Promises, which is an object that will return an asynchronous value. This allows a 'chaining' of calls, so it is possible to make a query to a database, and using the data retrieved to make another query to a different database, and as well as adding an intuitive way of catching errors using 'then' and 'catch' blocks. The implementation is shown in Figure 4.5 and an example of using the Promise query is shown in Figure 4.7.

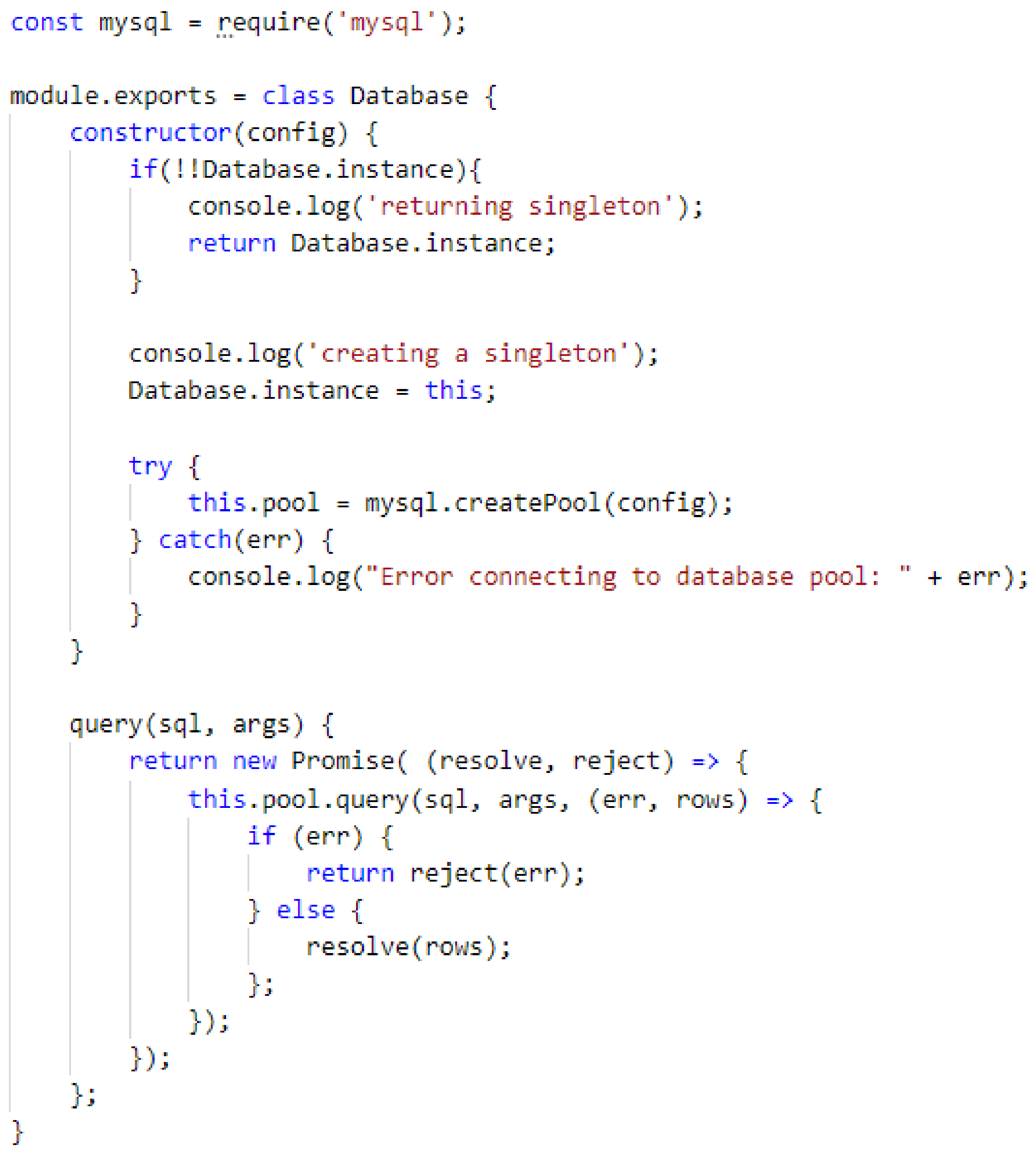
API-Construction - side effects of a template programming pattern.

4.6 Web App

The web app was constructed with the Model-View-Controller (MVC) architecture in mind. In Angular, components are used to create web pages. A 'component' consists of the following files, where 'myapp' is a placeholder name:

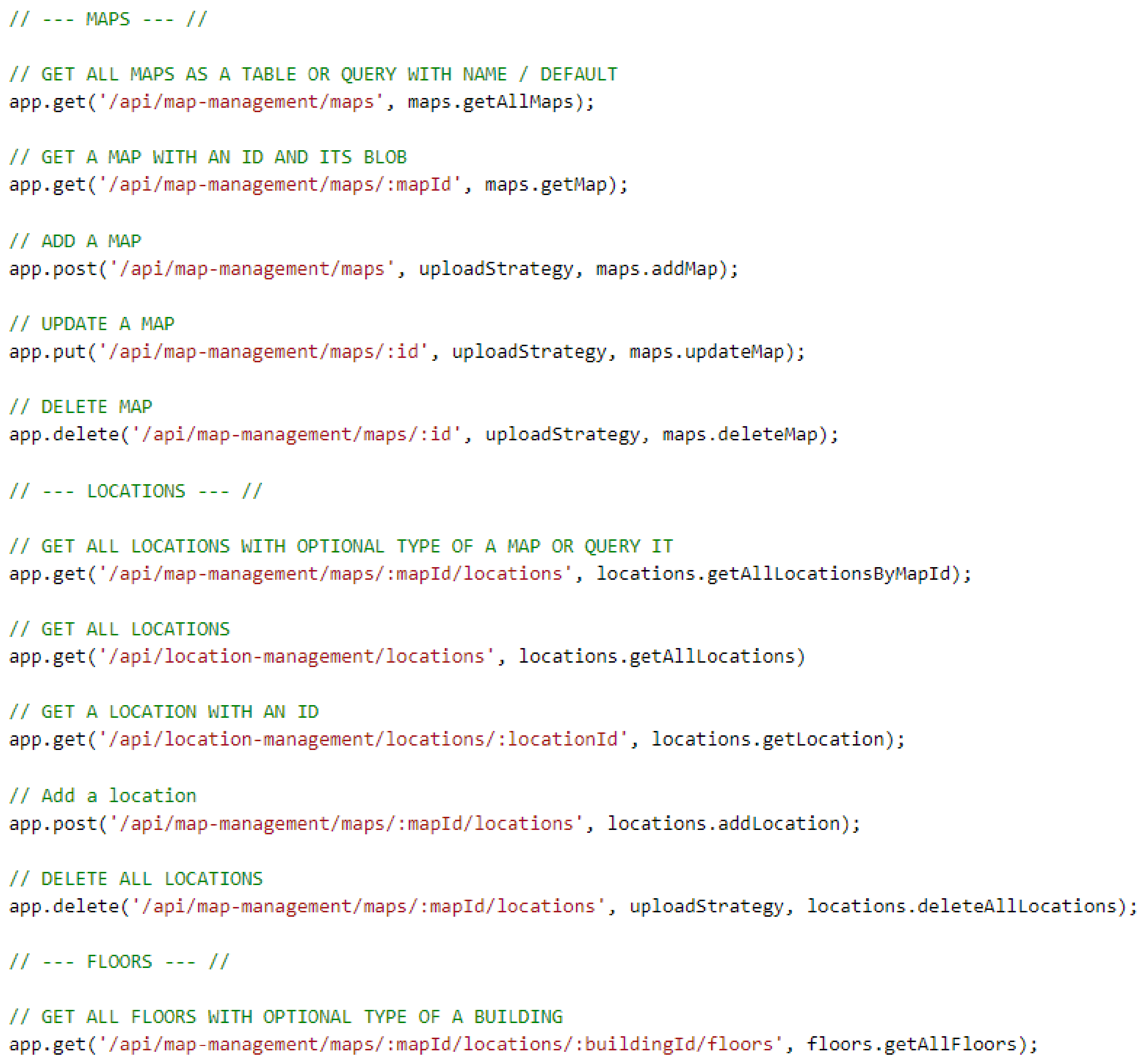
- myapp.ts
- myapp.spec.ts
- myapp.html
- myapp.css

A component in the Angular system implements both the view and the controller. A view determines how a page is displayed, and is implemented by the 'myapp.html' and 'myapp.css' files in a component; A controller controls how data flows into the view and operates on data,



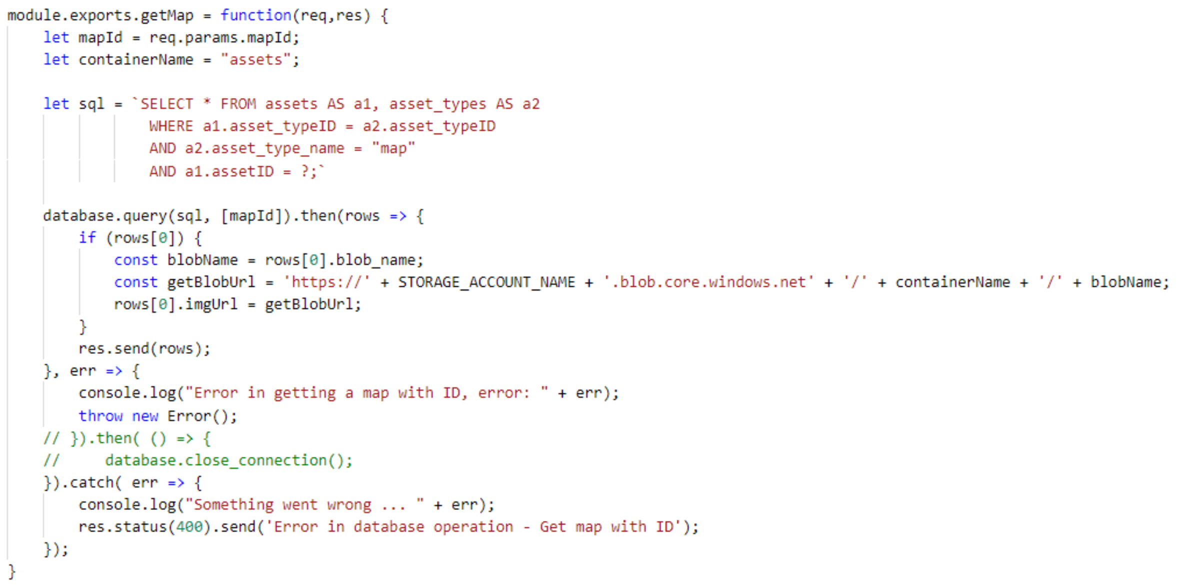
```javascript
const mysql = require('mysql');

module.exports = class Database {
    constructor(config) {
        if(!!Database.instance){
            console.log('returning singleton');
            return Database.instance;
        }

        console.log('creating a singleton');
        Database.instance = this;

        try {
            this.pool = mysql.createPool(config);
        } catch(err) {
            console.log("Error connecting to database pool: " + err);
        }
    }

    query(sql, args) {
        return new Promise( (resolve, reject) => {
            this.pool.query(sql, args, (err, rows) => {
                if (err) {
                    return reject(err);
                } else {
                    resolve(rows);
                };
            });
        });
    };
}
```

Figure 4.5: An excerpt of the database.js file in the server. It is implemented as a singleton, and creates a connection pool. It connects to a database based on database connection credentials from a config argument.



```
// --- MAPS --- //

// GET ALL MAPS AS A TABLE OR QUERY WITH NAME / DEFAULT
app.get('/api/map-management/maps', maps.getAllMaps);

// GET A MAP WITH AN ID AND ITS BLOB
app.get('/api/map-management/maps/:mapId', maps.getMap);

// ADD A MAP
app.post('/api/map-management/maps', uploadStrategy, maps.addMap);

// UPDATE A MAP
app.put('/api/map-management/maps/:id', uploadStrategy, maps.updateMap);

// DELETE MAP
app.delete('/api/map-management/maps/:id', uploadStrategy, maps.deleteMap);

// --- LOCATIONS --- //

// GET ALL LOCATIONS WITH OPTIONAL TYPE OF A MAP OR QUERY IT
app.get('/api/map-management/maps/:mapId/locations', locations.getAllLocationsByMapId);

// GET ALL LOCATIONS
app.get('/api/location-management/locations', locations.getAllLocations)

// GET A LOCATION WITH AN ID
app.get('/api/location-management/locations/:locationId', locations.getLocation);

// Add a location
app.post('/api/map-management/maps/:mapId/locations', locations.addLocation);

// DELETE ALL LOCATIONS
app.delete('/api/map-management/maps/:mapId/locations', uploadStrategy, locations.deleteAllLocations);

// --- FLOORS --- //

// GET ALL FLOORS WITH OPTIONAL TYPE OF A BUILDING
app.get('/api/map-management/maps/:mapId/locations/:buildingId/floors', floors.getAllFloors);
```

Figure 4.6: An excerpt of the server.js file in the server. This file defines the routes and connects them with the correct implementation methods from other imported files.



```
module.exports.getMap = function(req,res) {
    let mapId = req.params.mapId;
    let containerName = "assets";

    let sql = `SELECT * FROM assets AS a1, asset_types AS a2
                WHERE a1.asset_typeID = a2.asset_typeID
                AND a2.asset_type_name = "map"
                AND a1.assetID = ?;`

    database.query(sql, [mapId]).then(rows => {
        if (rows[0]) {
            const blobName = rows[0].blob_name;
            const getBlobUrl = 'https://' + STORAGE_ACCOUNT_NAME + '.blob.core.windows.net' + '/' + containerName + '/' + blobName;
            rows[0].imgUrl = getBlobUrl;
        }
        res.send(rows);
    }, err => {
        console.log("Error in getting a map with ID, error: " + err);
        throw new Error();
    // }).then( () => {
    //     database.close_connection();
    }).catch( err => {
        console.log("Something went wrong ... " + err);
        res.status(400).send('Error in database operation - Get map with ID');
    });
}
```

Figure 4.7: An excerpt of the maps.js file in the server. This file is one of the many files that define implementation for their respective behaviours. For example, this excerpt defines the getMap function that will be used by the GET routes defined in server.js.

which is the 'myapp.ts' files in a component; 'Model' are objects that hold data, which is implemented as a class with multiple properties in a '.ts' file under the 'classes' folder, but is not part of a component. These are imported into 'myapp.ts' to be used to hold data. Lastly, the '.spec.ts' file is a test file.

In Angular, it is possible to use a component within a component, and that will overlay the sub-component view on top of the main component view. This is useful for code reuse and extensibility. In the following example, I was trying to build a homepage which has multiple tabs (Figure3.3). The 'Summary' component contains the basic skeleton structure, and requires the data - 'title', 'dataSource' and 'displayedColumns' to be displayed. This component is used by other components - Anchors, Assets and Maps as the view and they each define the necessary data, i.e. 'title' defines the name of the page. 'dataSource' represents the rows of data retrieved from the database, where each main component will define their own way of retrieving data. and 'displayedColumns' defines the data columns to display. This architecture is shown in Figure 4.8. Each button wired has a defined route of 'title' + '/add' for the add button, and 'title' + '/edit' for the edit button.

For example, in the Maps component, 'maps.component.ts' retrieves data and stores it in variables (Figure 4.9). These variables are passed into the Summary component through 'maps.component.html' (Figure 4.10) using Angular's data binding process which allows variables used in '.ts' files to be used in the same component's '.html' files. In the routing section of Angular, shown in Figure 4.11, a 'MapsAddForm' component was created and wired to the 'maps/add' route. This method was chosen as the views for the different pages were similar

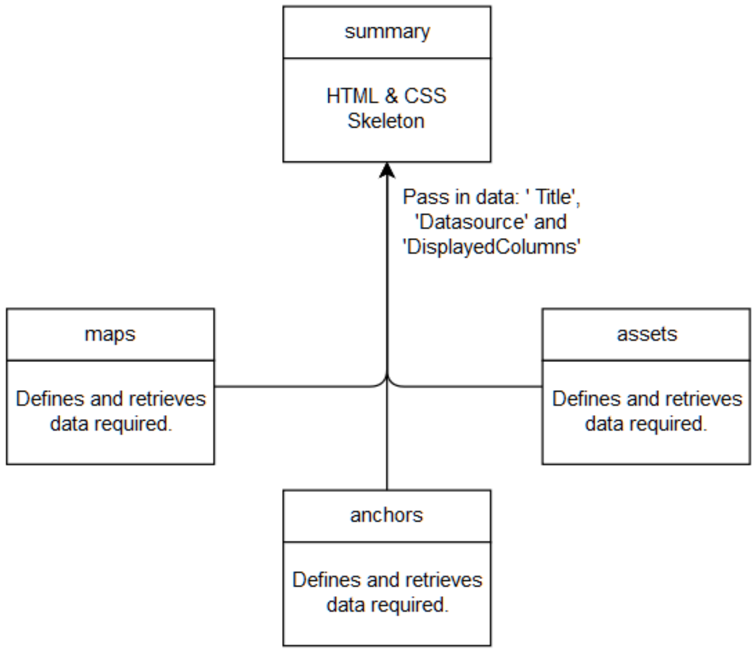
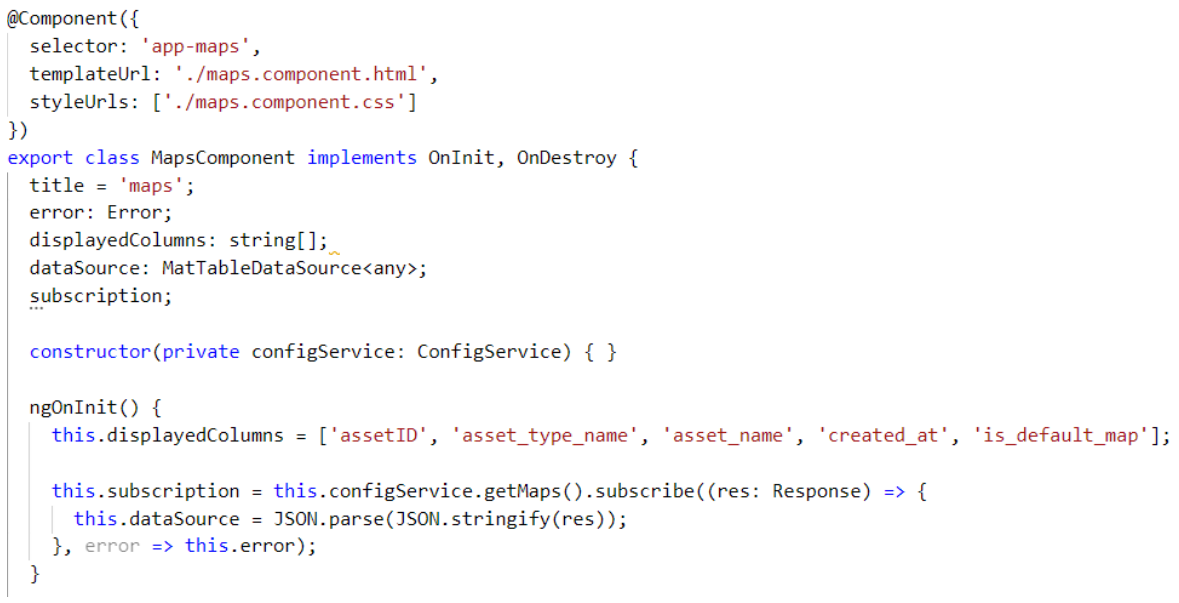
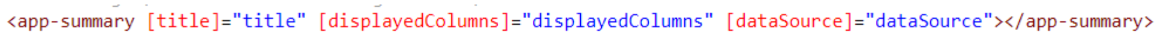

summary
HTML & CSS Skeleton
Pass in data: ' Title', 'Datasource' and 'DisplayedColumns'
maps
Defines and retrieves data required.
assets
Defines and retrieves data required.
anchors
Defines and retrieves data required.

Figure 4.8: Components in Angular to demonstrate component reuse. Maps, Anchors and Assets components retrieve data and pass in to the shared component - Summary to be displayed.



```
@Component({
  selector: 'app-maps',
  templateUrl: './maps.component.html',
  styleUrls: ['./maps.component.css']
})
export class MapsComponent implements OnInit, OnDestroy {
  title = 'maps';
  error: Error;
  displayedColumns: string[];
  dataSource: MatTableDataSource<any>;
  subscription;

  constructor(private configService: ConfigService) { }

  ngOnInit() {
    this.displayedColumns = ['assetID', 'asset_type_name', 'asset_name', 'created_at', 'is_default_map'];

    this.subscription = this.configService.getMaps().subscribe((res: Response) => {
      this.dataSource = JSON.parse(JSON.stringify(res));
    }, error => this.error);
  }
```

Figure 4.9: An excerpt from maps.component.ts, showing the variables needed to be assigned and how data was retrieved.



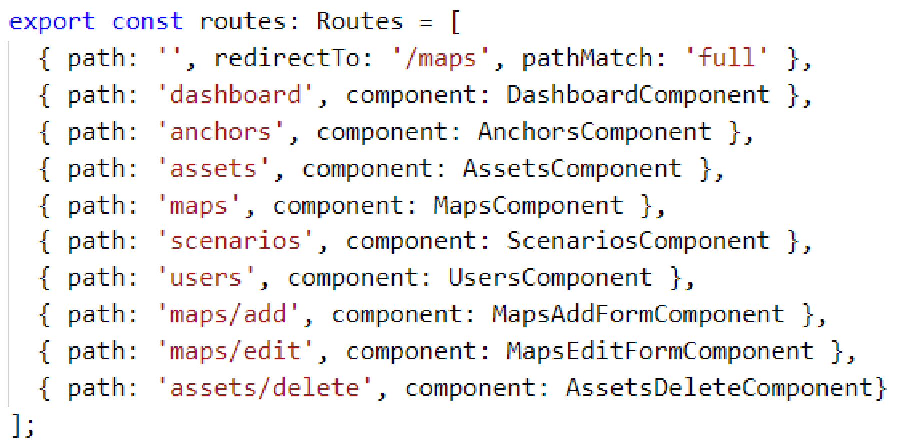
```
<app-summary [title]="title" [displayedColumns]="displayedColumns" [dataSource]="dataSource"></app-summary>
```

Figure 4.10: From maps.component.html, the line enables the use of the Summary component and passes in the values - Title, displayedColumns and dataSource.

except for the data required, hence a component can be reused many times by as many tabs needed. Every time a new tab is added, all that is needed to do is to create a main tab component, pass in the correct to the Summary component, create other needed components and wire the routes correctly.

The main components use 'config.service.ts' to make requests to the server. This file is known as a Service in Angular, and services are special in that there is only one instance of it throughout the project, behaving like a singleton. Services in Angular use Dependency Injection. It is a technique of providing objects with objects that they need (dependencies) from external sources instead of creating it themselves. In this case for example, the Maps component would not need to know how to connect to a database and what API request to make to retrieve the data needed, it just needs to delegate this responsibility to someone else, i.e. config.service.ts and a call a function from this service. This allows the component to have less responsibility, and allows easier unit testing as the service can be mocked.

In Angular, HTTP requests follow the Observer design pattern. It returns an observable (subject) which observers can subscribe to and receive data from. It is beneficial as the CPU can run other tasks while it waits for a response. When a response is received, a dedicated callback function will be run. An example is shown in Figure 4.9 where the function 'getMaps()' is subscribed to.



```
export const routes: Routes = [
  { path: '', redirectTo: '/maps', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'anchors', component: AnchorsComponent },
  { path: 'assets', component: AssetsComponent },
  { path: 'maps', component: MapsComponent },
  { path: 'scenarios', component: ScenariosComponent },
  { path: 'users', component: UsersComponent },
  { path: 'maps/add', component: MapsAddFormComponent },
  { path: 'maps/edit', component: MapsEditFormComponent },
  { path: 'assets/delete', component: AssetsDeleteComponent}
];
```

Figure 4.11: Routes in app-routing.module.ts, to be used by the app to route to different page views defined by components.

For the website, Maps, Anchors, and Assets components were created. As Maps is the only one that can add data, and has an interesting method instead of only filling a form, the flow will be described. The process flow diagram is shown in Figure 4.12. A drawable canvas was created to create ’locations’ on an uploaded map, and the goal is that every time a location is pressed, a pop up will appear to fill in details. For this drawable canvas, instead of creating as many components as there are locations drawn, a single component - ’LocationForm’ is used along with a list of ’Location’ objects. Each location object has a property - ’relativePosition’ on the map, and every time the user clicks on a location on the map, this list is iterated through, checking whether a location object is within the region clicked. When this is true, the location object reference is passed into the created Location-Form component, and the information entered is saved into the same reference. This saves the need of creating multiple components at run time, improving speed and efficiency.

4.7 Mobile App

4.7.1 Database

Connection is done through ’database.cs’, which is also a singleton, implemented with a private constructor and a public ’Instance’ property. Each request passes in JSON data and retrieves JSON data, which is then stored in objects. The passing and use of JSON data allows the app to be more portable to other databases in the future if needed.

4.7.2 Augmented Reality Implementation

The app was built starting from the samples given by ASA documentation [26]. The main files that controls the state of the system are ’AdminCrudAnchorsController.cs’ and ’User-

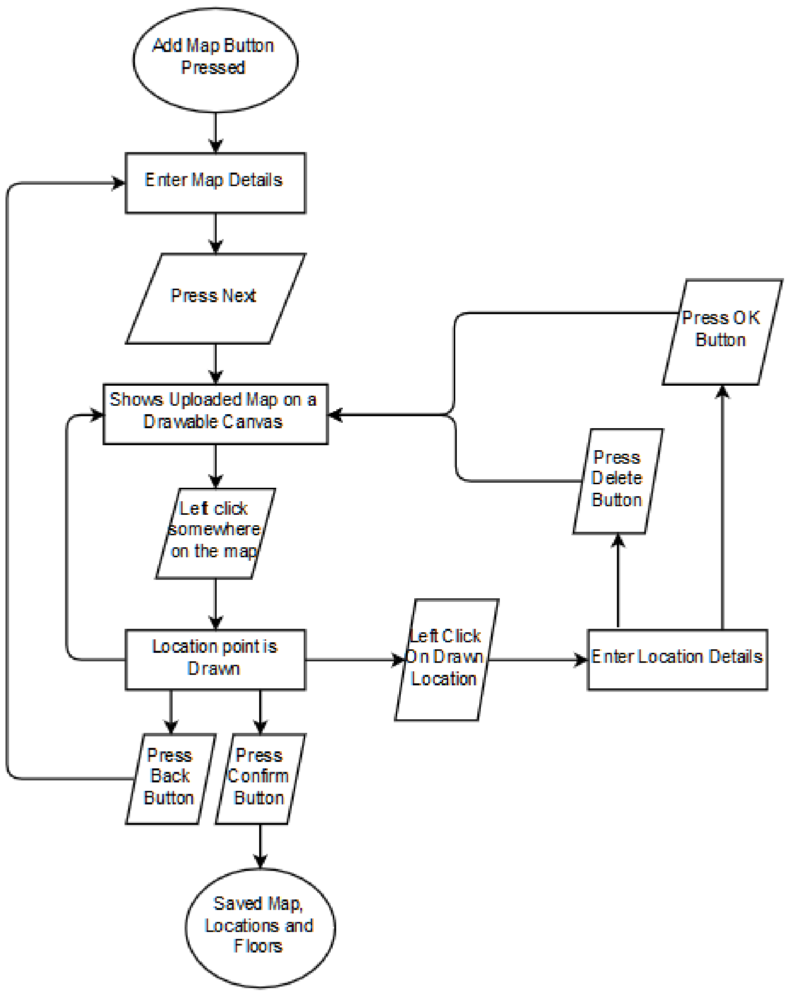
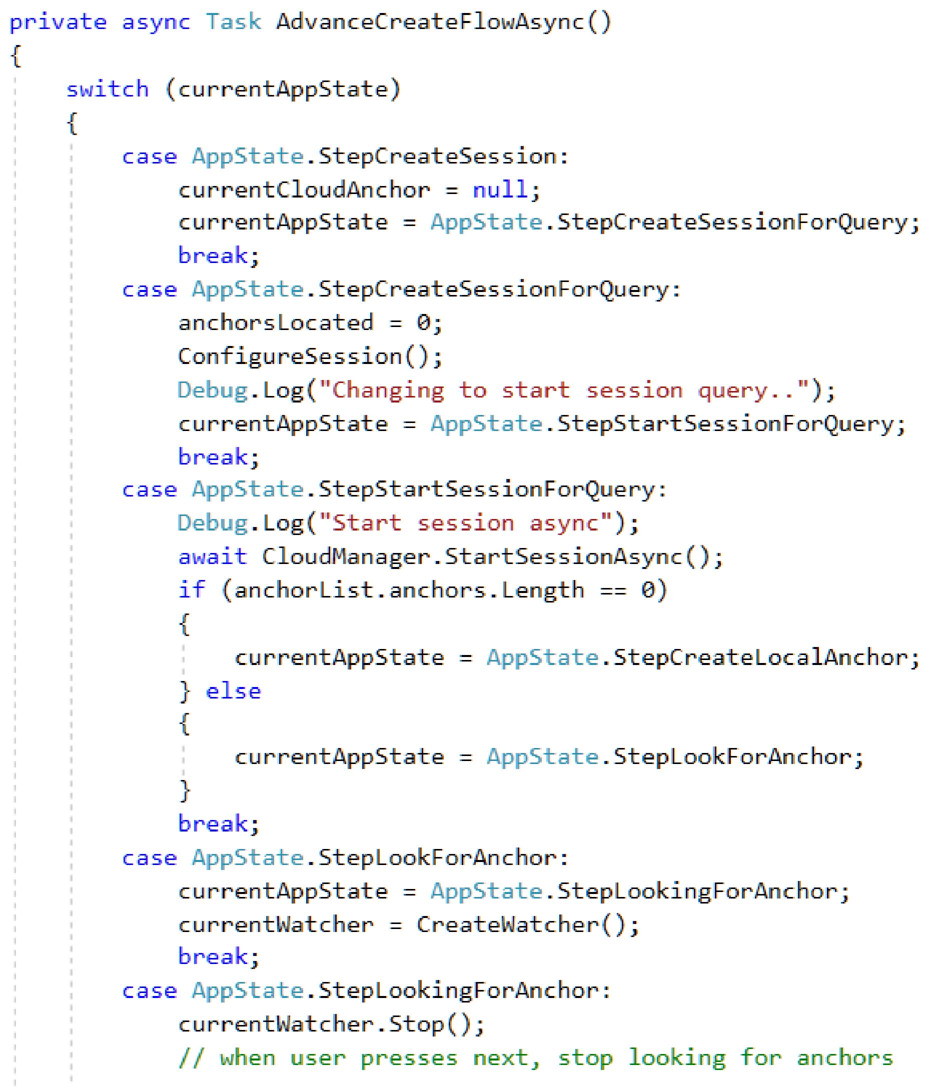


Figure 4.12: Add Map flowchart. A user will go through the steps above to add a map and attach locations to the map.

LocateAnchorsController.cs'. Both of these files inherit the base class 'ArAppScriptBase.cs'. The latter file implements important functionality such as the creation of an anchor, saving of an anchor and etc. It utilises the Template programming pattern. For example, in 'ArAppScriptBase.cs' the function 'SaveCurrentObjectAnchorToCloudAsync()' calls the virtual method 'OnSaveCloudAnchorSuccessfulAsync()' when it successfully saves an anchor. However, this method has no implementation in the file, but is implemented in its derived classes.

In the controller files, they implement the Strategy pattern. They hold all the possible states of the current system, and delegate functions depending on the state through the use of 'AdvanceCreateFlowAsync' or 'AdvanceLocateFlowAsync' functions. Taking the example of 'AdvanceCreateFlowAsync' (Figure 4.13), the function defines a list of switch-case statements to run different implementations depending on the current app state. This function runs



```
private async Task AdvanceCreateFlowAsync()
{
    switch (currentAppState)
    {
        case AppState.StepCreateSession:
            currentCloudAnchor = null;
            currentAppState = AppState.StepCreateSessionForQuery;
            break;
        case AppState.StepCreateSessionForQuery:
            anchorsLocated = 0;
            ConfigureSession();
            Debug.Log("Changing to start session query..");
            currentAppState = AppState.StepStartSessionForQuery;
            break;
        case AppState.StepStartSessionForQuery:
            Debug.Log("Start session async");
            await CloudManager.StartSessionAsync();
            if (anchorList.anchors.Length == 0)
            {
                currentAppState = AppState.StepCreateLocalAnchor;
            } else
            {
                currentAppState = AppState.StepLookForAnchor;
            }
            break;
        case AppState.StepLookForAnchor:
            currentAppState = AppState.StepLookingForAnchor;
            currentWatcher = CreateWatcher();
            break;
        case AppState.StepLookingForAnchor:
            currentWatcher.Stop();
            // when user presses next, stop looking for anchors
```

Figure 4.13: An excerpt of 'AdminCrudController.cs'.

whenever a 'next' button is pressed.

To record input, some code from object.manipulation.cs from Google [27] was used as a starting point. It involves taking touch inputs and registering them as specific Gestures, e.g. a tap gesture and a drag gesture. 'Manipulator' files are files that affect the movement of an object using these gestures, e.g. files such as TranslationManipulator and SelectionManipulator were used, the former uses the drag gesture to move an object, and the latter uses a tap gesture to register a selection.

4.7.3 Dynamic Loading of Assets

The flow for a user to upload an asset into the database is to:

1. Create 3d Asset with a Unity supported 3D modelling software.
2. Import into Unity,
3. Create a prefab (dragging it into the Project editor, and bringing it into the Prefab folder)
4. Delete the object in the project editor.
5. Select the prefab created.
6. On the right hand bottom corner, next to 'AssetBundles', choose a name name of file.
7. On the top, select 'Assets' > 'Build and Upload Assetbundles'. Wait awhile for Unity to complete.
8. (Optional) Delete prefab file.

The script 'Editor/CreateAssetBundles.cs' was created to create a tab named 'Build and Upload Assetbundles' and to facilitate the build process for AssetBundles and uploading. Once it is done, it deletes the built files. The AssetBundle is now uploaded to blob storage and a database record is created with a blob URL. This URL was used to load the AssetBundle and retrieve the asset for the mobile app.

4.8 Unimplemented Work

This section is dedicated to work that was designed, but not implemented. However, some design choices may be beneficial, and future users who wish to extend the app may be able to use this.

4.8.1 Treasure Hunt System

A scenario can be created for each treasure hunt game. Each scenario is a series of tasks. Each task contains some information and a clue about what to find. In the back-end, the mobile app should continuously search for all anchors, but only display the anchors that are attached to this Task. Attaching anchors to the task could include by attaching locations instead, so that all anchors at the specific location are also attached to the same task. Assets can also be uploaded so that each task can have a series of assets, including non-anchored ones. Those can be displayed to the user in a modal, right after an anchored asset is scanned.

4.8.2 Navigation System

ASA allows for wayfinding. The difference between wayfinding and navigating is that wayfinding acts like a compass: it tells you the direction of the desired location, but doesn't tell you the route to take [25]. ASA tells you whether there are nearby anchors, and how far away they are. For this to work, ASA requires anchors to be 'connected' in the same AR session. Connection can be either creating all anchors in the same session, or load one and create anchors in the same session.

By having all anchors connected, the problem is now two fold:

1. How to know start point?

   The start point is something the user will have to scan an anchor in the room to begin with. An anchor can be created for everything, including entrances so this compromise might not have a huge impact.

2. How to know end point?

   An anchor can be attached to a 'Location' from the map system that was created. The user can select the location they need to go to, and the app will search for the anchor that is attached to the location.

   Hence, by selecting a start anchor, and selecting a destination anchor, the app can perform a wayfind between these two anchors.

To improve on this wayfinding by turning it into a turn-by-turn navigation, 3 new problems need to be solved:

4. How to know directions to turn?

   Create 'Locations' on turning points in corridors. Generate a connected graph based on these Location nodes. Distances between nodes can be inputted as a value by the website user. The network of nodes, along with the starting and end points is just the Travelling Salesman problem. To find the shortest route, a potential algorithm to use is Dijkstra's algorithm.

5. How to know which floors to take?

   Generate a node at every junction, i.e. stairs and elevators, and connect the appropriate nodes.

6. How to know current position?

   A possible solution would be to create empty anchors, not displayable to mobile users at all of these nodes, and allow ASA to perform wayfinding in between nodes of the path. A possible issue may be that this is too intensive and the phone may run slow, hence would require testing.

Chapter 5

Testing

5.1 Overview

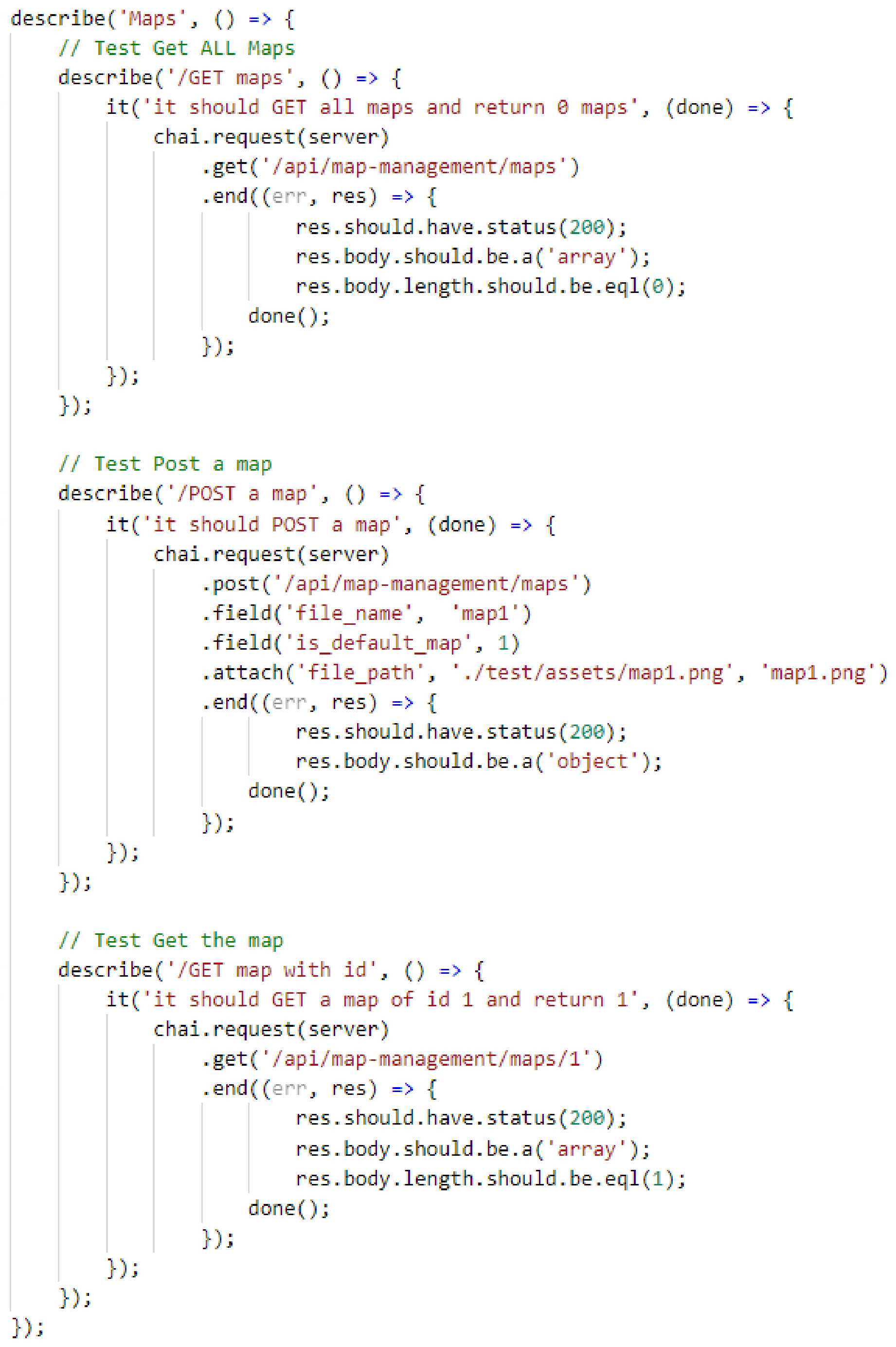
Unit tests, integration tests, system tests and acceptance tests were carried out. Unit tests are defined as single tests for a single unit only without external dependencies, and integration tests are defined as tests with dependencies. System testing involves testing the ability of a system to perform under load and acceptance testing involves whether system functionalities are accepted by the end users. Try and Catch blocks were used throughout the project to catch exceptions and help in the debugging process.

5.2 Database

Two databases were used, one is for production and one for testing & development. This was useful because if the production database is used, some unnecessary data may be stored after the tests either due to insufficient post-tests database clean up, or due to errors generated in the testing process.

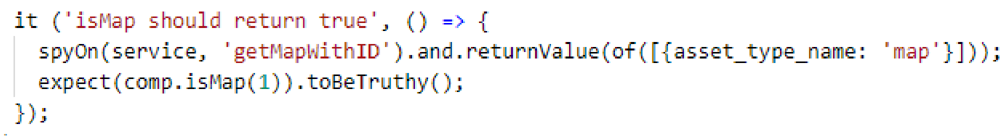
5.3 Server

Mocha is a Javascript framework which allows for asynchronous testing, it provides an environment to use assertion libraries to test the code. An assertion library allows the testing framework to assert whether a unit test returns the expected result. The assertion library used is Chai. In testing for an API server, only tests made are integration tests as they involve making requests to the database server. An example to test the map functionality is shown in Figure 5.1. A lot of testing involved manual testing, using Postman to make and verify API requests on a local server before deploying to the production server.



```
describe('Maps', () => {
    // Test Get ALL Maps
    describe('/GET maps', () => {
        it('it should GET all maps and return 0 maps', (done) => {
            chai.request(server)
                .get('/api/map-management/maps')
                .end((err, res) => {
                        res.should.have.status(200);
                        res.body.should.be.a('array');
                        res.body.length.should.be.eql(0);
                    done();
                });
        });
    });

    // Test Post a map
    describe('/POST a map', () => {
        it('it should POST a map', (done) => {
            chai.request(server)
                .post('/api/map-management/maps')
                .field('file_name',  'map1')
                .field('is_default_map', 1)
                .attach('file_path', './test/assets/map1.png', 'map1.png')
                .end((err, res) => {
                        res.should.have.status(200);
                        res.body.should.be.a('object');
                    done();
                });
        });
    });

    // Test Get the map
    describe('/GET map with id', () => {
        it('it should GET a map of id 1 and return 1', (done) => {
            chai.request(server)
                .get('/api/map-management/maps/1')
                .end((err, res) => {
                        res.should.have.status(200);
                        res.body.should.be.a('array');
                        res.body.length.should.be.eql(1);
                    done();
                });
        });
    });
});
```

Figure 5.1: Test script written for the server.



```
it ('isMap should return true', () => {
  spyOn(service, 'getMapWithID').and.returnValue(of([{asset_type_name: 'map'}]));
  expect(comp.isMap(1)).toBeTruthy();
});
```

Figure 5.2: Excerpt of a test file in Angular utilising Jasmine's spyOn to isolate tests from service implementation.

5.4 Web App

To create Unit and Integration tests, Jasmine and Karma testing frameworks were used. A benefit of using Angular's dependency injection framework for services is that functions can be isolated from services' functions. As shown in Figure 5.2, it uses Jasmine's spyOn function to make the function return a value that is expected, and the test case does not need to know the underlying implementation of 'getMapWithId'.

Some integration tests were created to check for routes to different pages. Besides that, there were a lot of manual testing using Chrome with Developer Tools enabled on a local server with 'console.log' before deploying onto the production server.

5.5 Mobile App

Tests were mostly done with Unity's play mode with console logs. It emulates the app on the Unity application, and objects in the Unity scene can be interacted with to quickly make iterations.

To test for AR functionality, this involved building the app to the mobile device and testing on it, as it was the only way to use the AR camera. Console logs were read through Android Logcat from the Unity Editor.

Testing for asset bundles involved this as well. The build process was tested first for correct uploads to the database and blob storage, and then tested whether it would load properly in the app.

5.6 System Testing

Throughout the duration of the project, the database, on the basic service plan, stayed at an average of 11.8% usage, only increasing by 0.02% at maximum. Blob storage has increased up to 5gb, but as all Azure services are easily scalable by requesting more cloud resources, the app should be able to scale easily to a large number of users.

Through manual testing, the web app and server were tested to be quick enough for viewing different web pages and retrieving & accessing data respectively. For the mobile app, there is a small delay in the retrieval of data for loading of locations during app startup, but not very significant.

5.7 Acceptance Testing

For UI, interviews were done initially with GOSH supervisors and the PGME committee for feedback on the mock up, hence the final product as long as it looks similar should be acceptable. Throughout the implementation process, usage of the app and website were demonstrated to supervisors to receive feedback.

Chapter 6

Conclusion and Project Evaluation

6.1 Overview

6.2 Achievements

6.2.1 Project Achievements

The goals outlined in Introduction were all achieved:

1. A website was built and connected to a backend.

2. An android application connecting to the same backend was built.

3. A pipeline involving Assetbundles was set up to upload and load 3D models. The pipeline for maps were set up using the website.

4. AR was used in the development of the app.

From the requirements, the project has achieved around 60% of requirements completed. The important problems to solve were a method of navigation, a method of using AR and a treasure hunt system built in. The first one was mostly achieved, except support for location info including opening and closing times were not done, this was due a lack of time. However, a pop up to display this location info was created. The second one was also almost done, the main issue remaining was to improve 3D asset editing (e.g. by adding scaling, rotation, and elevation) and have the database support this. The Treasure Hunt system, was also not done due to a lack of time. The lack of time resulted from not formulating the requirements well enough and splitting time between many parts of the project.

However, what has been achieved is a maps system that users can immediately use to view maps and locations, using it as a interactive, pocket map tool. GOSH staff will be able to upload their own maps and update them easily. Besides that, AR has been implemented, and users can place their own anchors and assets. Tests can be made to check whether ASA can perform well enough in the hospital. Furthermore, a lot of the backbone needed to create this ecosystem has been done, and if someone were to continue this project, they would be able to focus solely on creating rich features.

6.2.2 Personal Achievements

Personal achievements were that I have 1. Learnt what is Unity and how to code in C# 2. Learnt how to implement augmented reality in Unity for Android. 3. Learnt how to do dynamic loading through asset bundles. 4. Learnt how to do Unit Testing and Integration Testing. 5. Learnt how to set up API Server in ExpressJS to make requests from database and storage. 6. Hosted web apps, database and storage in the cloud using Azure Services. 7. Learnt how to create drawings on website using the Canvas element 8. Learnt different design patterns and their use cases.

6.3 Critical Evaluation

Maps system can be used. GOSH can upload the maps and create the locations they want, and users can view the maps on the app easily. The AR system can be further improved by having better asset manipulation code, but still can be used to place assets, albeit it will require a decent sized model before loading into the project, which may require trial and error. Server has a decent defined set of routes that will enable easy calls,

6.4 Future Steps

To improve on the app, the following steps are recommended:

1. **Maps**. Create location info on the app by adding extra fields to the location-form on the website, extra fields in the database, and to display these fields on the mobile app. A column for each day of the week can be created, as well as a column for telephone numbers and another column for website address.

2. **Maps**. Create the 'search' bar that allows a user to enter the name of the location they want.

3. **AR**. Improve on the asset manipulator code to include scaling, elevation and rotation. The 'objectmanipulation.cs' code from Google may help in this.

4. **Collection**. Add in a collection system to make it more fun. Add in a feedback animation when someone finds an asset.

5. **Navigation**. Some suggestions can be found in the Design and Implementation section to improve navigation on the app.

6.5 Final Thoughts

Despite not narrowing requirements well enough and setting out to do too much, I still learnt an overwhelming amount from the project. I was able to learn how to connect mobile, app, and website together through a REST-like server, to a database and a separate storage, as well as learning how to implement AR. The project now has a good backbone for future improvements.

Bibliography

[1] TP026B, Rev. Rational Unified Process. Available from:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf [Accessed on 01/09/2019]

[2] M. Beedle, M. Devos, Y. Sharon, K. Schwaber, and J. Sutherland, vol. 4, N. Harrison, Ed. Boston: Addison-Wesley, 1999, pp. 637-651. Available from:
https://www.scruminc.com/wp-content/uploads/2014/05/Scrum-A-Pattern-Language-for-Software-Development.pdf
[Accessed on 01/09/2019]

[3] Workflow organisation software, Trello. Available from: https://trello.com/ [Accessed on 01/09/2019]

[4] What is Kanban? Atlassian. Available from: https://www.atlassian.com/agile/kanban [Accessed on 01/09/2019]

[5] Ronald T.Azuma. A Survey On Augmented Reality. Presence: Teleoperators and Virtual Environments., vol. 6, no. 4, pp. 355-385, 1997. Available from: https://www.mitpressjournals.org/doi/pdfplus/10.1162/pres.1997.6.4.355 [Accessed on 01/09/2019]

[6] Kelly M. Microsoft secures $480 million HoloLens contract from US Army. The Verge, 2018. Available from: https://www.theverge.com/2018/11/28/18116939/microsoft-army-hololens-480\linebreak-million-contract-magic-leap [Accessed on 27/08/2019]

[7] MacPhedran S. Augmented Manufacturing: The Big Six HoloLens Use Cases for Manufacturers.SMITH, 2018. Available from: https://blog.smith.co/2018/augmented-manufacturing [Accessed on 31/08/2019]

[8] Liberati N. Phenomenology, Pokmon Go, and Other Augmented Reality Games. Human Studies, Volume 41, Issue 2, pp 211232, 2018. Available from: https://link.springer.com/article/10.1007/s10746-017-9450-8 [Accessed on 31/08/2019]

[9] Milgram P., Takemura H., Utsumi A., Kishino F. Augmented Reality: A class of displays on the reality-virtuality continuum. Proceedings of Telemanipulator and Telepresence Technologies. pp. 235134. 1994. Available from: `http://etclab.mie.utoronto.ca/publication/1994/Milgram_Takemura_SPIE1994.pdf` [Accessed on 28/08/2019]

[10] Rogers K, Frommel J, Breier L, Celik S, Kramer H, Kreidel S, Brich J, Riemer V, Schrader C. Mobile augmented reality as an orientation aid: a scavenger hunt prototype. International Conference on Intelligent environments (IE), pp 172175, 2015. Available from: `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.100/institut/mitarbeiterbereiche/frommel/Publications/ie15_unirallye.pdf` [Accessed on 29/08/2019]

[11] Lu, Y., Chao, J. T., & Parker, K. HUNT: Scavenger hunt with augmented reality. Interdisciplinary Journal of Information, Knowledge, and Management, 10, 21-35, 2015. Available from: `http://www.ijikm.org/Volume10/IJIKMv10p021-035Lu1580.pdf` [Accessed on 26/08/2019]

[12] Vuforia. Available from: `https://www.ptc.com/en/products/augmented-reality` [Accessed on 01/09/2019]

[13] ARCore, Google. Available from: `https://developers.google.com/ar/` [Accessed on 02/09/2019]

[14] ARKit, Apple. Available from: `https://developer.apple.com/documentation/arkit` [Accessed on 29/08/2019]

[15] ARFoundation, Unity. Available from: `https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html` [Accessed on 29/08/2019]

[16] Azure Spatial Anchors, Microsoft. Available from: `https://azure.microsoft.com/en-gb/services/spatial-anchors/` [Accessed on 29/08/2019]

[17] Azure Blob Storage, Microsoft. Available from: `https://azure.microsoft.com/en-gb/services/storage/blobs/` [Accessed on 29/08/2019]

[18] Web App Mock-Up, Figma. Available from: `https://www.figma.com/file/QHg6xxs6GGOjBnSqA6QEtfAg/AR-Treasure-Hunt-Website?node-id=0\%3A1`

[19] Mobile App Mock-Up, Figma. Available from: `https://www.figma.com/file/FxUuFrEJzaaYOnDzM7hb3FWo/AR-Treasure-Hunt-App?node-id=0\%3A1`

[20] Three Tier Architectures. IBM. 2019. Available from: `https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/covr_3-tier.html` [Accessed on 28/08/2019]

[21] Connolly, T. M., & Begg, C. E. Database systems: A practical approach to design, implementation, and management. 2002. Harlow, England: Addison-Wesley.

[22] MEAN Stack. IBM. 2019. Available from: `https://www.ibm.com/cloud/learn/mean-stack-explained` [Accessed on 29/08/2019]

[23] Fielding R T. Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis. University of California; 2000.

[24] RESTful Web Services - Caching. Tutorialspoint. Available from: `https://www.tutorialspoint.com/restful/restful_caching.htm` [Accessed on 30/08/2019].

[25] Anchor relationships and way-finding in Azure Spatial Anchors. Microsoft, 2019. Available from: `https://docs.microsoft.com/en-us/azure/spatial-anchors/concepts/anchor-relationships-way-finding` [Accessed on 29/08/2019]

[26] Azure Spatial Anchors Samples, Microsoft. Available from: `https://github.com/Azure/azure-spatial-anchors-samples` [Accessed on 02/09/2019]

[27] Object Manipulation Sample Code, Google. Available from: `https://developers.google.com/ar/develop/unity/tutorials/object-manipulation-sample` [Accessed on 01/09/2019]

Appendix A

System Manual

1. Github Repository URL

   `https://github.com/haowjern/UCL-Final-Year-Project-Hospital-Induction`

2. Web App URL

   `https://hospital-induction-website.azurewebsites.net`

3. Server URL

   `https://app-treasure-hunt-server.azurewebsites.net`

4. API Server Routes

   Refer to section C.4

5. MySQL Database Connection parameters

   - host: ’`app-treasure-hunt-database.mysql.database.azure.com`’,
   - user: ’haowjern@app-treasure-hunt-database’
   - password: ’lG7cV95J’,
   - database: ’app_mysql_test’ (for test database) or ’app_mysql’ (for production database),
   - port: 3306,
   - ssl: BaltimoreCyberTrustRoot.crt.pem

6. Azure Blob Storage Connection parameters

   - accountName: ’apptreasurehunt’
   - accountKey:
     ’N6Kw8xToZ8JEhTrfil59Vtc/bVt7Fnu5Lh4Ha3d+kxJGwT9e
     lb5euINQ8paQ6j9xokRVPEaO9Fek46PKzliR0g==’

7. Azure Spatial Anchors Connection parameters

    - Account Id: 33c2db82-fb57-4e29-8512-4cfe80dc1ea5
    - Account Key: wBfntfzrBldnCH6Z4gl0ZOMdc0qMB6QpBejnPtr55WQ=
    - Resource: 'https://sts.mixedreality.azure.com',

8. Azure Database For MySQL Set Up Guidelines

    When a client (the server) is accessing the database (this includes making server calls), the IP address that the client has needs to be included in Azure Database For MySQL. To do this, head into Azure Portal ¿ Select database ¿ Connecition Security ¿ Add Client IP.

    Hence for local testing of server, do take note to include the ip address of your local host.

Appendix B

User Manual

B.1 Web App

URL: `https://app-treasure-hunt-website.azurewebsites.net/`

To add a map, follow Figure 4.12.

B.1.1 Notes

- To edit a map and to change the file, you will need to delete the map first and and upload a new one.
- When drawing locations on the Add Map page, the page needs to be zoomed out until there is no scrolling so that drawing is accurate.

B.2 Mobile App

- Can't delete anchor because need to delete from app, ASA specifies this.

B.3 Uploading Asset Bundles

To create and upload asset bundles:

1. Select 3D model of your choice and ensures Unity supports this.
2. Load 3D model into any unity scene.
3. Drag 3D model from the scene into the folder 'Prefab' on the left hand side under Project to create a prefab.
4. On the right hand bottom corner, select AssetBundles and select a name.

5. Select 'Assets' > 'Build and Upload Asset Bundles' from the top menu bar.
6. Wait for awhile...
7. Once Unity is done, you have uploaded! Now you can use your asset in the app.
8. (Optional) Delete the 3D model that was previously created in the scene.
9. (Optional) Delete the prefab that was previously created in the Prefab folder.

Appendix C

Supporting Documents and Diagrams

C.1 Requirements List

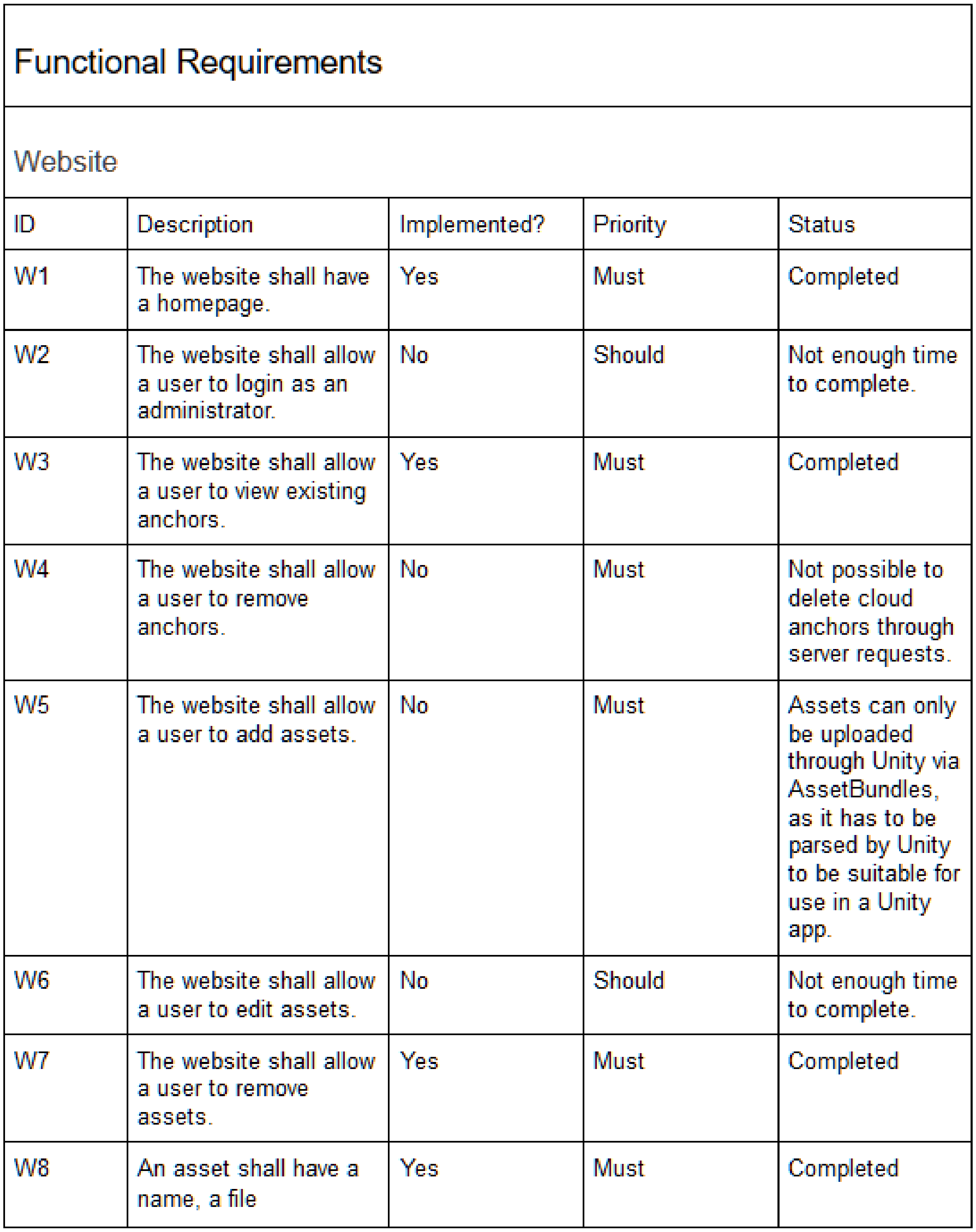
C.2 Detailed Use Cases

C.3 Mock Ups

To view mock-ups, please create a Figma account and access the links at [18] and [19] and press the play button (denoted with a right-facing arrow) on the top right hand corner.

C.4 API Server Routes

Please refer to Table C.21 for the list of routes. To make a request requires the Base Address + Route. Base Address is `https://app-treasure-hunt-server.azurewebsites.net`. Hence, a request for example is `https://app-treasure-hunt-server.azurewebsites.net/api/map-management/maps`.

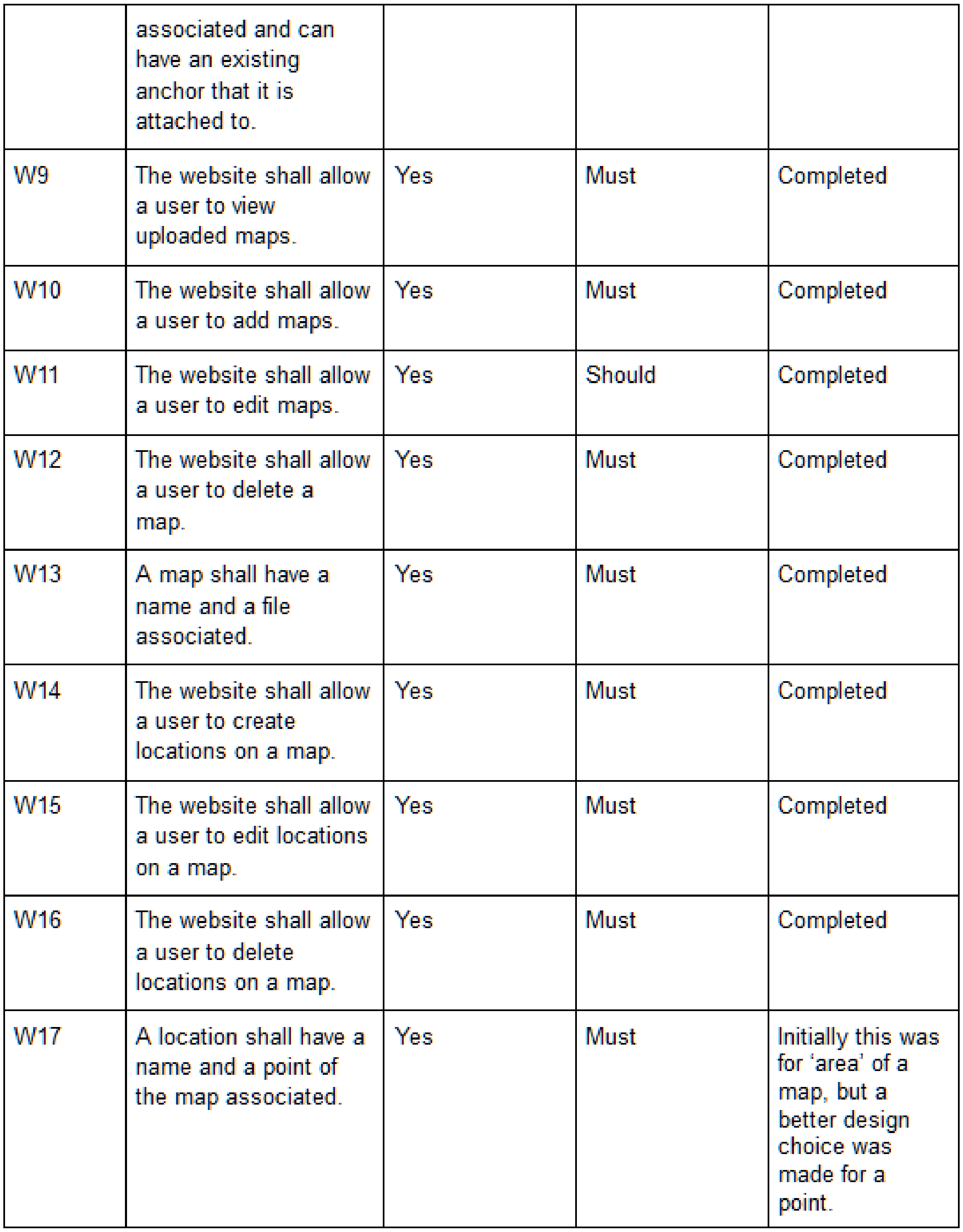


Functional Requirements

Website

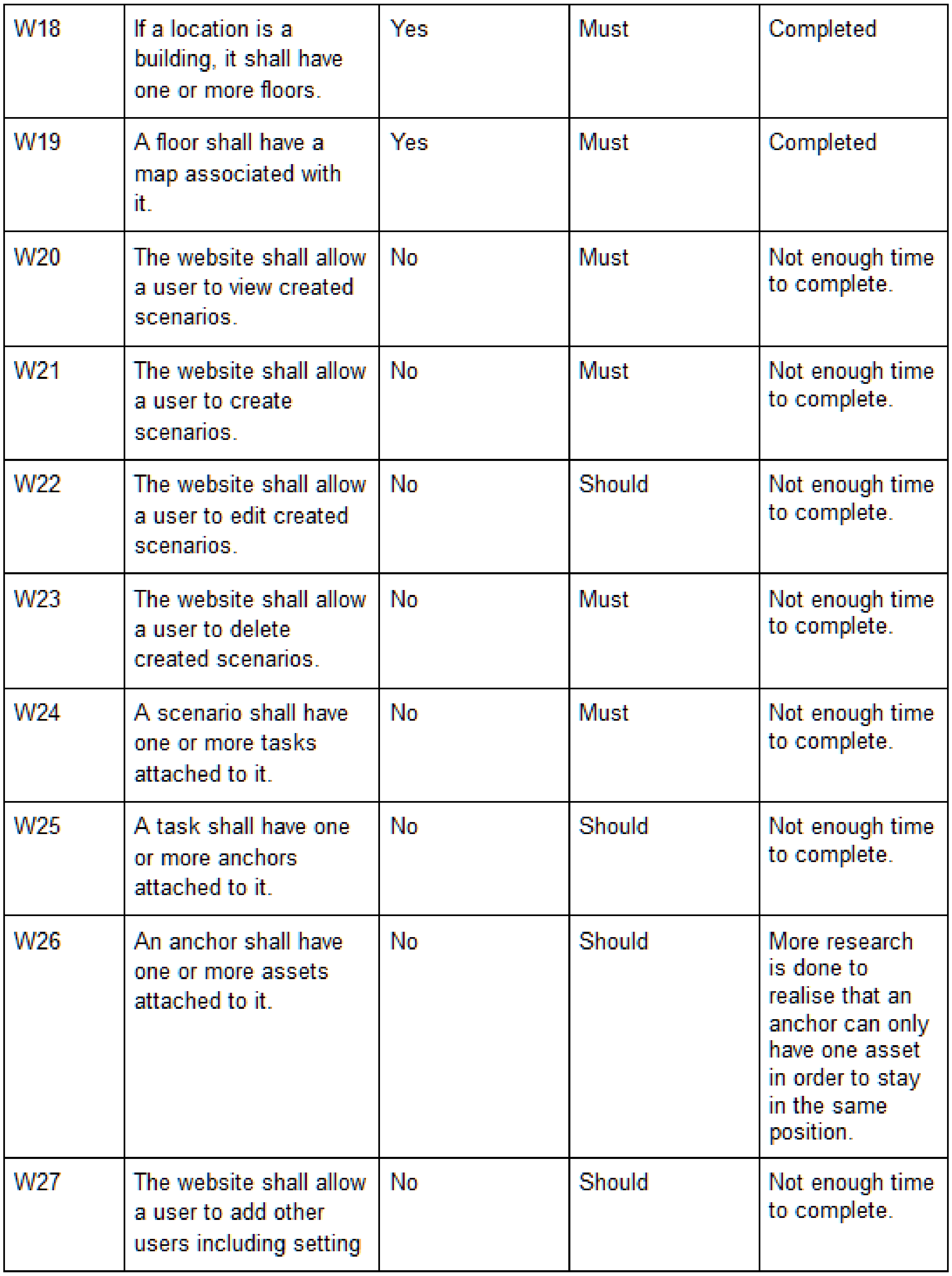
ID	Description	Implemented?	Priority	Status
W1	The website shall have a homepage.	Yes	Must	Completed
W2	The website shall allow a user to login as an administrator.	No	Should	Not enough time to complete.
W3	The website shall allow a user to view existing anchors.	Yes	Must	Completed
W4	The website shall allow a user to remove anchors.	No	Must	Not possible to delete cloud anchors through server requests.
W5	The website shall allow a user to add assets.	No	Must	Assets can only be uploaded through Unity via AssetBundles, as it has to be parsed by Unity to be suitable for use in a Unity app.
W6	The website shall allow a user to edit assets.	No	Should	Not enough time to complete.
W7	The website shall allow a user to remove assets.	Yes	Must	Completed
W8	An asset shall have a name, a file	Yes	Must	Completed

Table C.1: Requirements page 1.



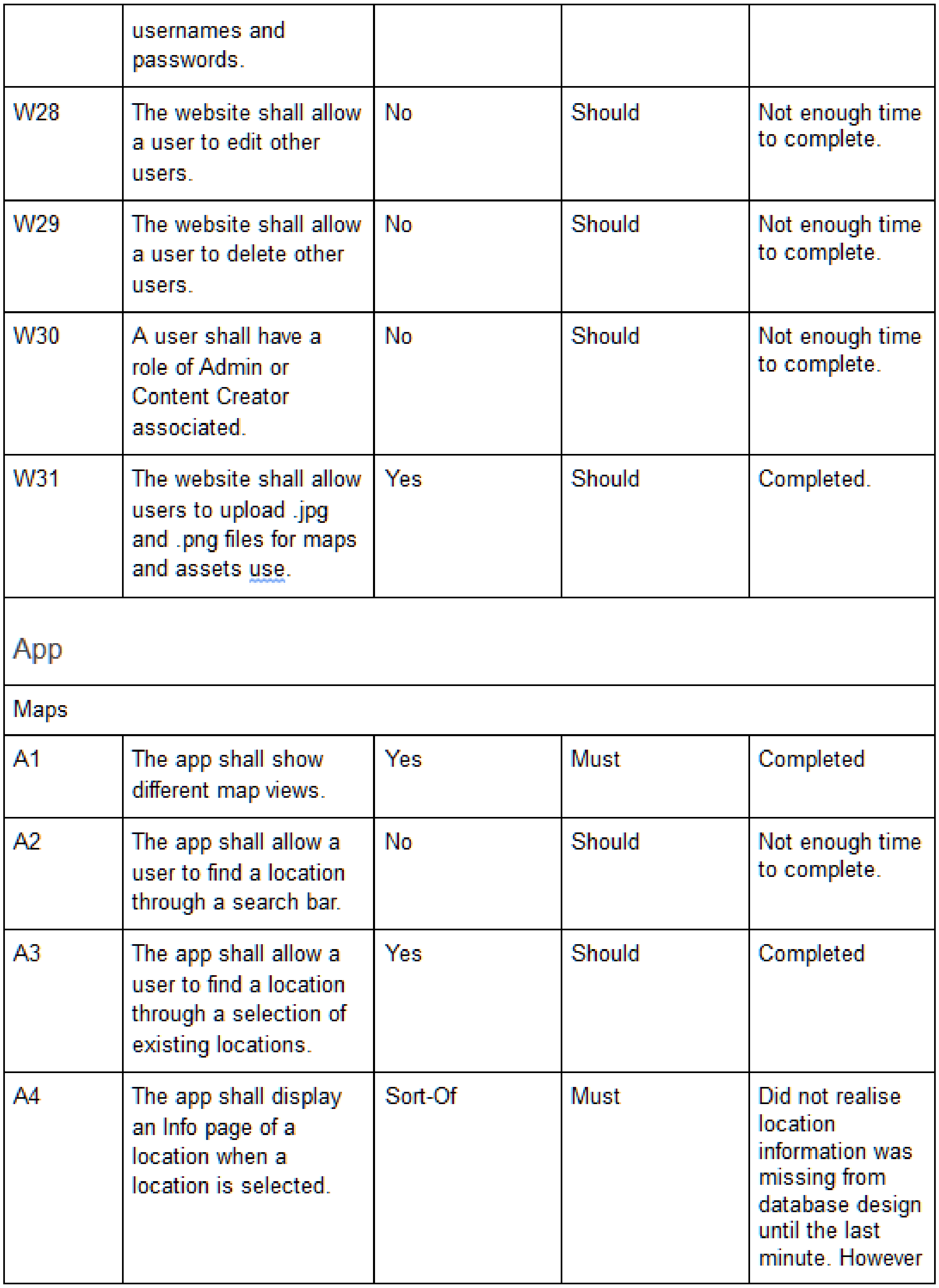
	associated and can have an existing anchor that it is attached to.			
W9	The website shall allow a user to view uploaded maps.	Yes	Must	Completed
W10	The website shall allow a user to add maps.	Yes	Must	Completed
W11	The website shall allow a user to edit maps.	Yes	Should	Completed
W12	The website shall allow a user to delete a map.	Yes	Must	Completed
W13	A map shall have a name and a file associated.	Yes	Must	Completed
W14	The website shall allow a user to create locations on a map.	Yes	Must	Completed
W15	The website shall allow a user to edit locations on a map.	Yes	Must	Completed
W16	The website shall allow a user to delete locations on a map.	Yes	Must	Completed
W17	A location shall have a name and a point of the map associated.	Yes	Must	Initially this was for 'area' of a map, but a better design choice was made for a point.

Table C.2: Requirements page 2.



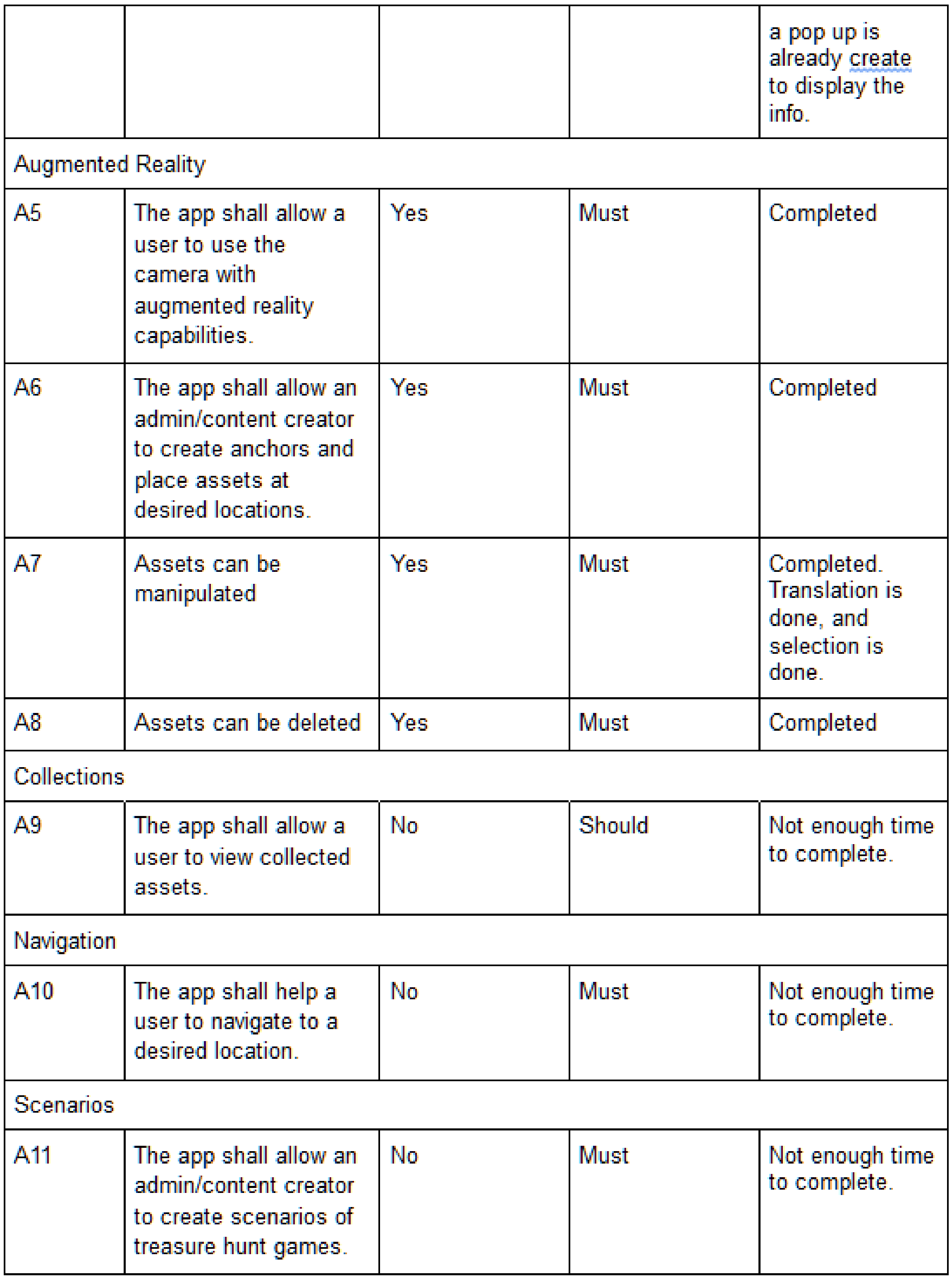
W18	If a location is a building, it shall have one or more floors.	Yes	Must	Completed
W19	A floor shall have a map associated with it.	Yes	Must	Completed
W20	The website shall allow a user to view created scenarios.	No	Must	Not enough time to complete.
W21	The website shall allow a user to create scenarios.	No	Must	Not enough time to complete.
W22	The website shall allow a user to edit created scenarios.	No	Should	Not enough time to complete.
W23	The website shall allow a user to delete created scenarios.	No	Must	Not enough time to complete.
W24	A scenario shall have one or more tasks attached to it.	No	Must	Not enough time to complete.
W25	A task shall have one or more anchors attached to it.	No	Should	Not enough time to complete.
W26	An anchor shall have one or more assets attached to it.	No	Should	More research is done to realise that an anchor can only have one asset in order to stay in the same position.
W27	The website shall allow a user to add other users including setting	No	Should	Not enough time to complete.

Table C.3: Requirements page 3..



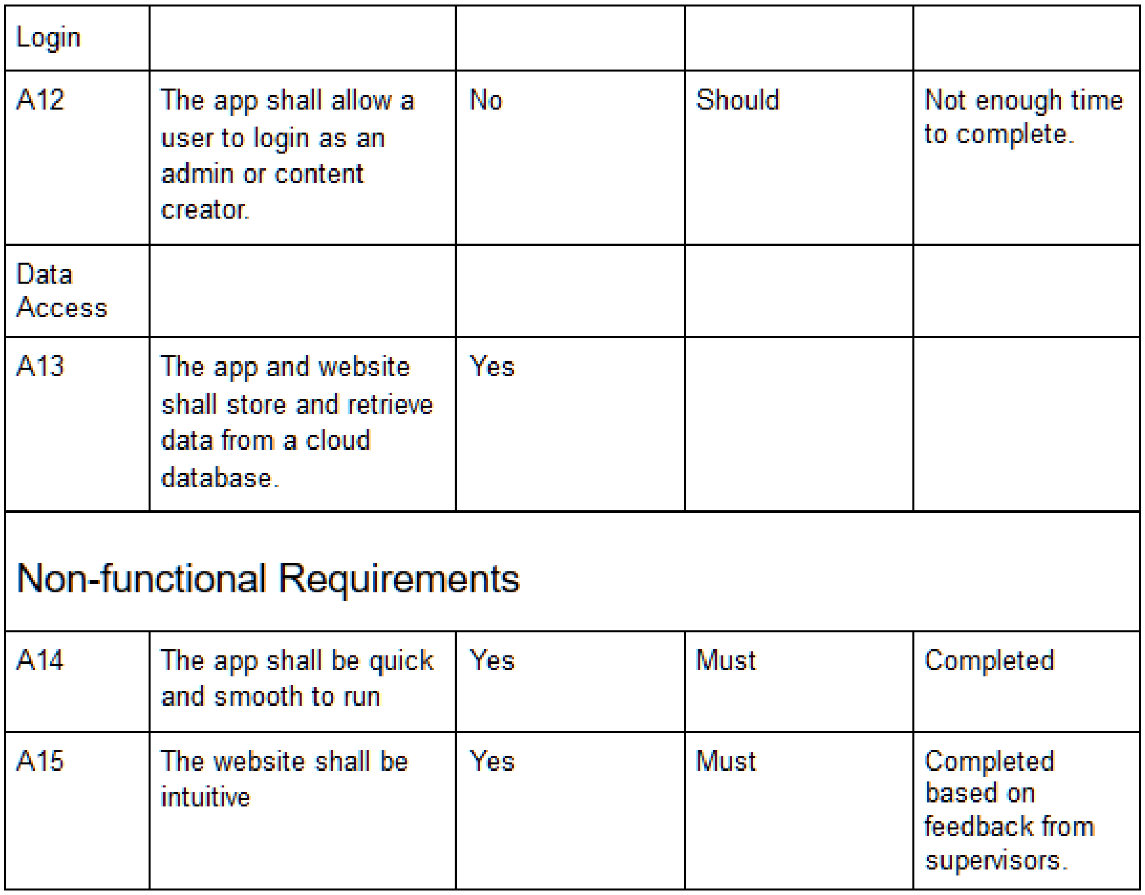
	usernames and passwords.			
W28	The website shall allow a user to edit other users.	No	Should	Not enough time to complete.
W29	The website shall allow a user to delete other users.	No	Should	Not enough time to complete.
W30	A user shall have a role of Admin or Content Creator associated.	No	Should	Not enough time to complete.
W31	The website shall allow users to upload .jpg and .png files for maps and assets use.	Yes	Should	Completed.
App				
Maps				
A1	The app shall show different map views.	Yes	Must	Completed
A2	The app shall allow a user to find a location through a search bar.	No	Should	Not enough time to complete.
A3	The app shall allow a user to find a location through a selection of existing locations.	Yes	Should	Completed
A4	The app shall display an Info page of a location when a location is selected.	Sort-Of	Must	Did not realise location information was missing from database design until the last minute. However

Table C.4: Requirements page 4.



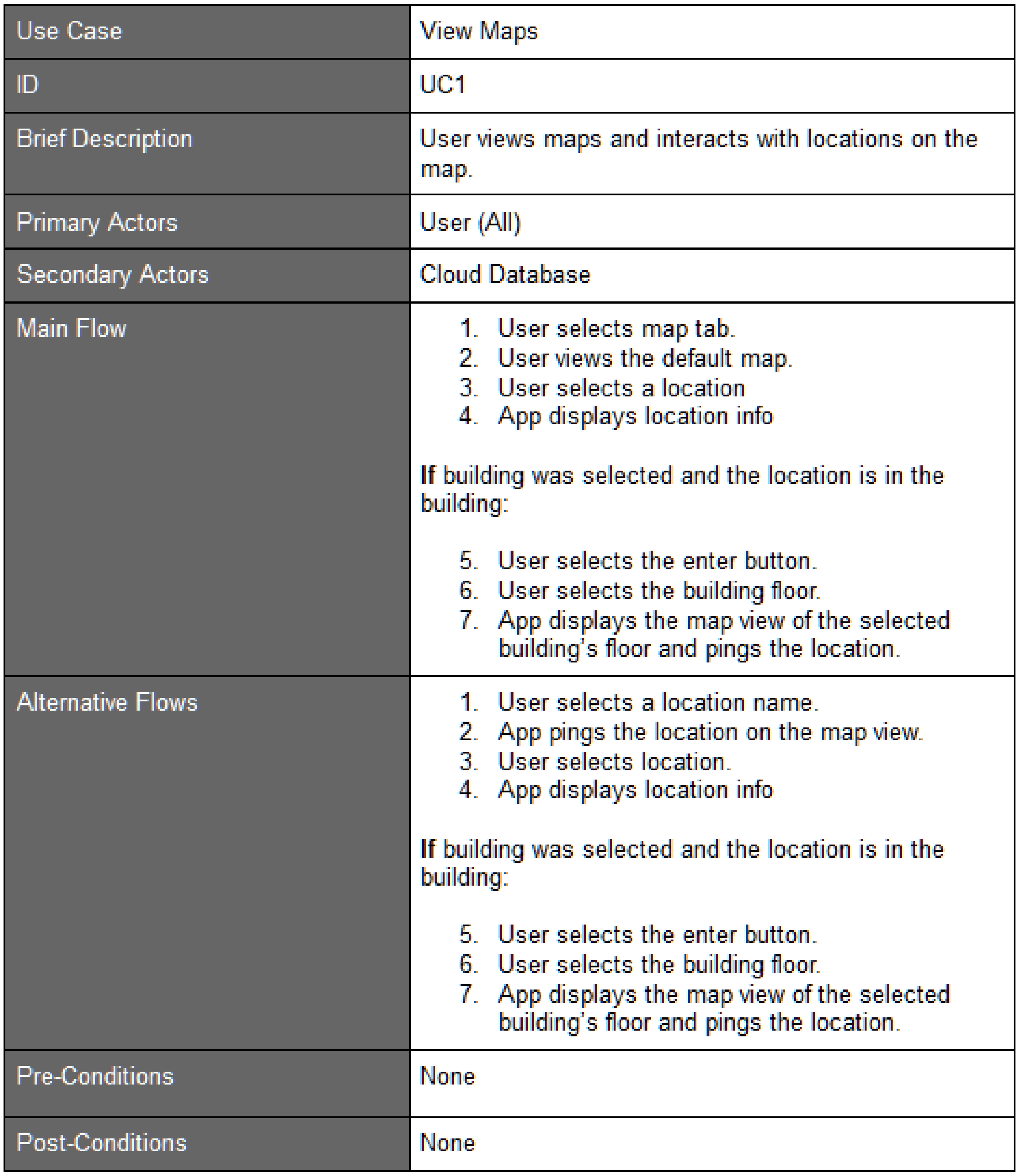
				a pop up is already create to display the info.
Augmented Reality				
A5	The app shall allow a user to use the camera with augmented reality capabilities.	Yes	Must	Completed
A6	The app shall allow an admin/content creator to create anchors and place assets at desired locations.	Yes	Must	Completed
A7	Assets can be manipulated	Yes	Must	Completed. Translation is done, and selection is done.
A8	Assets can be deleted	Yes	Must	Completed
Collections				
A9	The app shall allow a user to view collected assets.	No	Should	Not enough time to complete.
Navigation				
A10	The app shall help a user to navigate to a desired location.	No	Must	Not enough time to complete.
Scenarios				
A11	The app shall allow an admin/content creator to create scenarios of treasure hunt games.	No	Must	Not enough time to complete.

Table C.5: Requirements page 5.



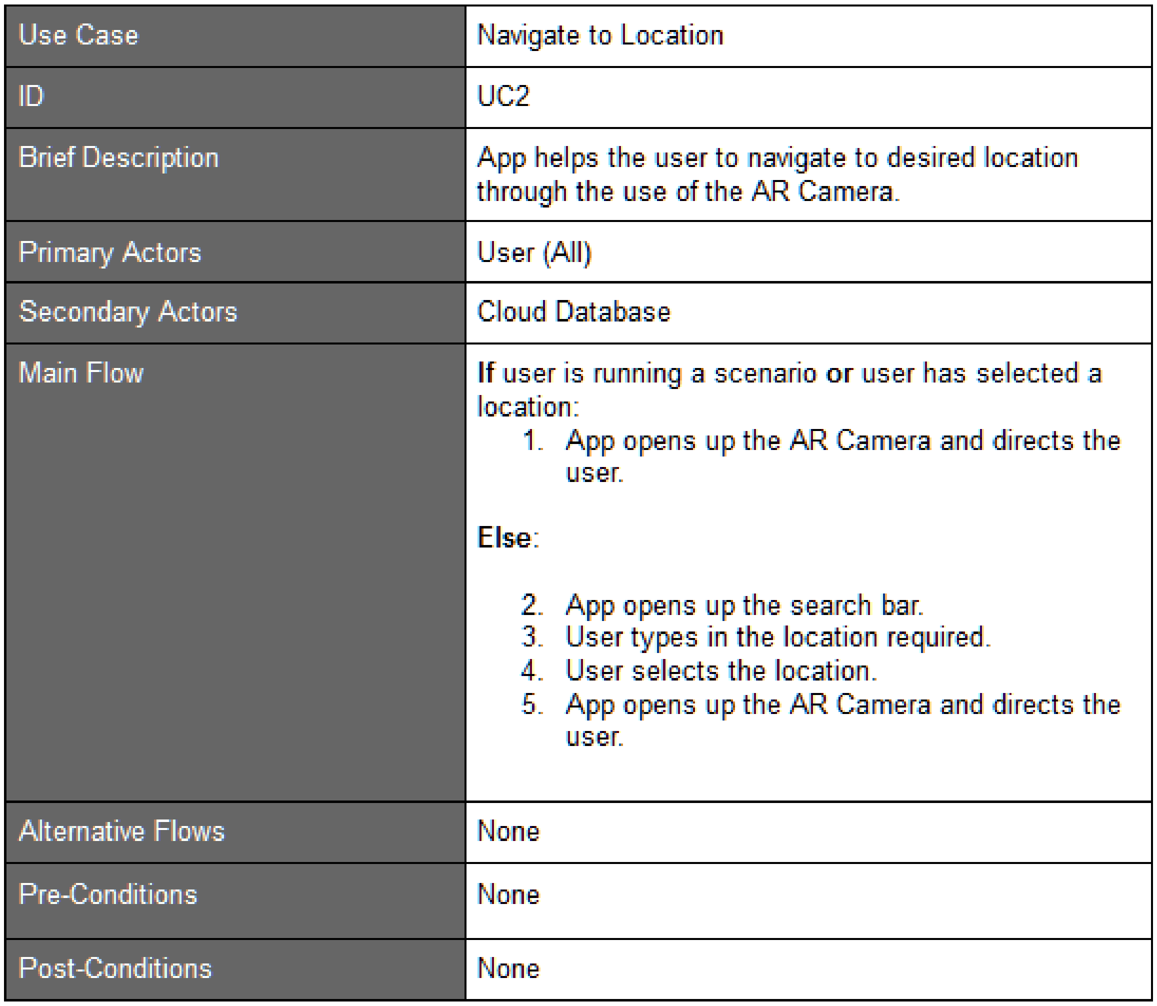
Login				
A12	The app shall allow a user to login as an admin or content creator.	No	Should	Not enough time to complete.
Data Access				
A13	The app and website shall store and retrieve data from a cloud database.	Yes		
Non-functional Requirements				
A14	The app shall be quick and smooth to run	Yes	Must	Completed
A15	The website shall be intuitive	Yes	Must	Completed based on feedback from supervisors.

Table C.6: Requirements page 6.



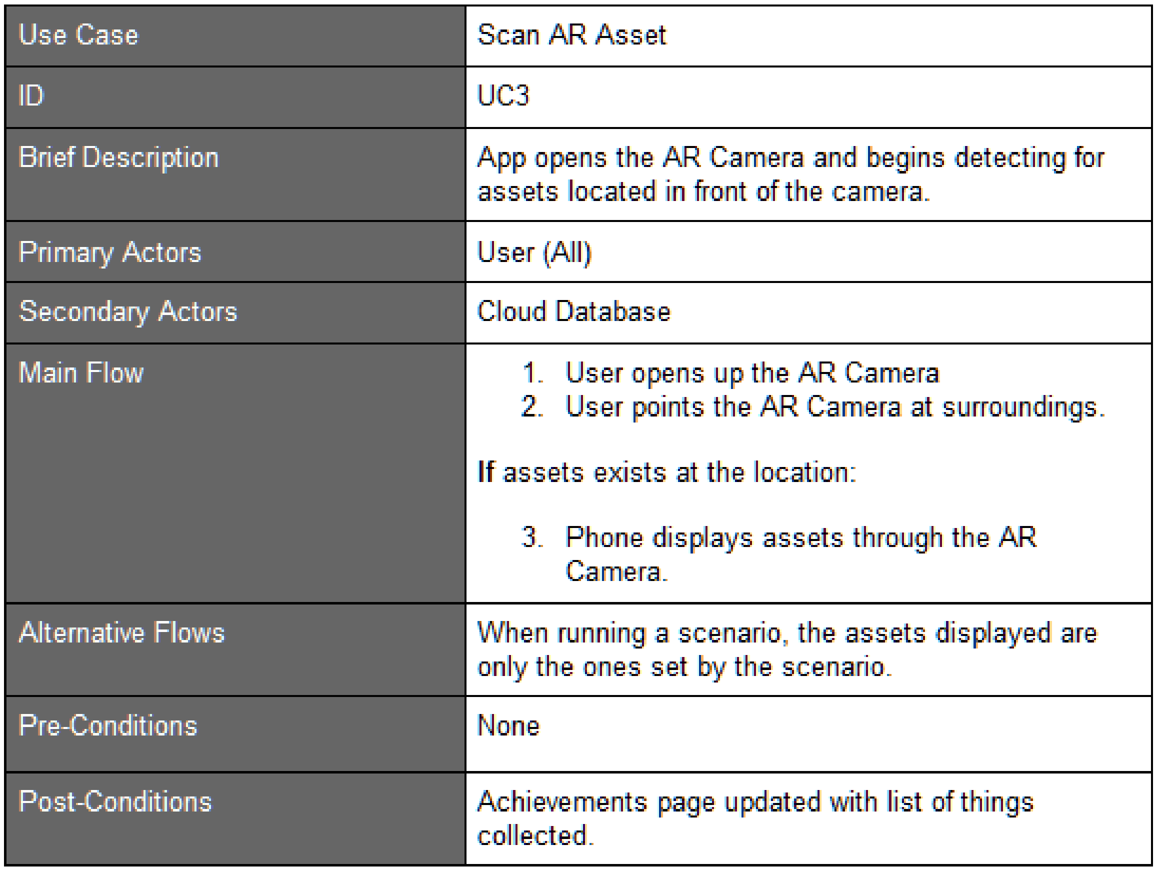
Use Case	View Maps
ID	UC1
Brief Description	User views maps and interacts with locations on the map.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	1. User selects map tab. 2. User views the default map. 3. User selects a location 4. App displays location info **If** building was selected and the location is in the building: 5. User selects the enter button. 6. User selects the building floor. 7. App displays the map view of the selected building's floor and pings the location.
Alternative Flows	1. User selects a location name. 2. App pings the location on the map view. 3. User selects location. 4. App displays location info **If** building was selected and the location is in the building: 5. User selects the enter button. 6. User selects the building floor. 7. App displays the map view of the selected building's floor and pings the location.
Pre-Conditions	None
Post-Conditions	None

Table C.7: Detailed Use Case: View Maps.



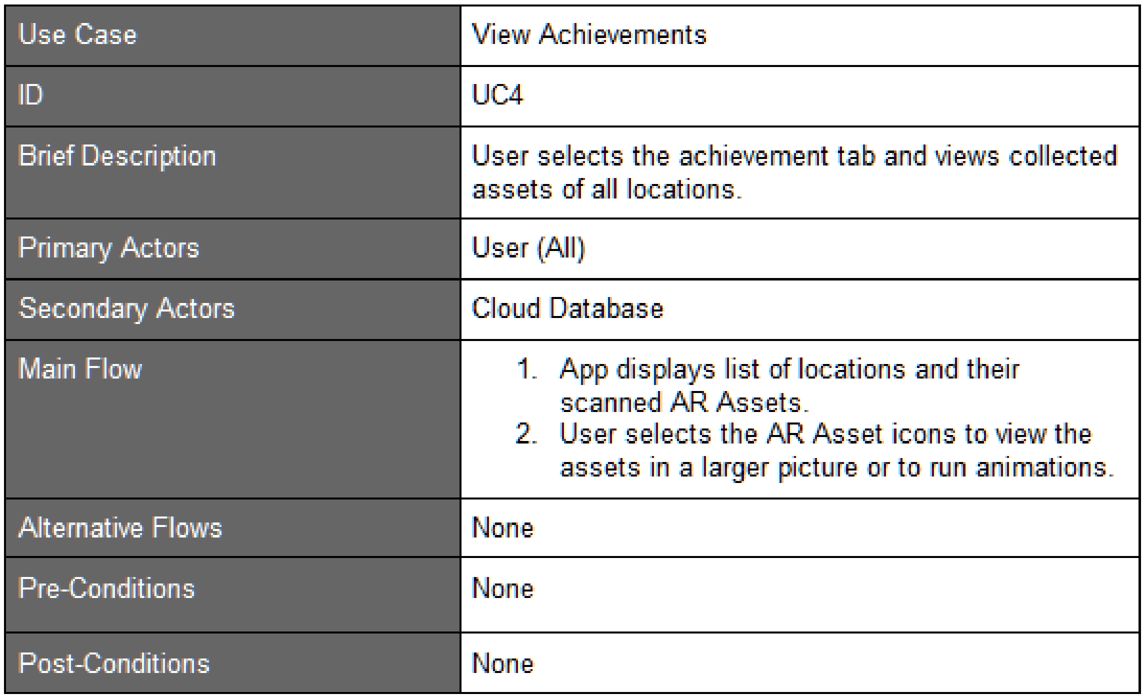
Use Case	Navigate to Location
ID	UC2
Brief Description	App helps the user to navigate to desired location through the use of the AR Camera.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	**If** user is running a scenario **or** user has selected a location: 1. App opens up the AR Camera and directs the user. **Else**: 2. App opens up the search bar. 3. User types in the location required. 4. User selects the location. 5. App opens up the AR Camera and directs the user.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.8: Detailed Use Case: Navigate to Location



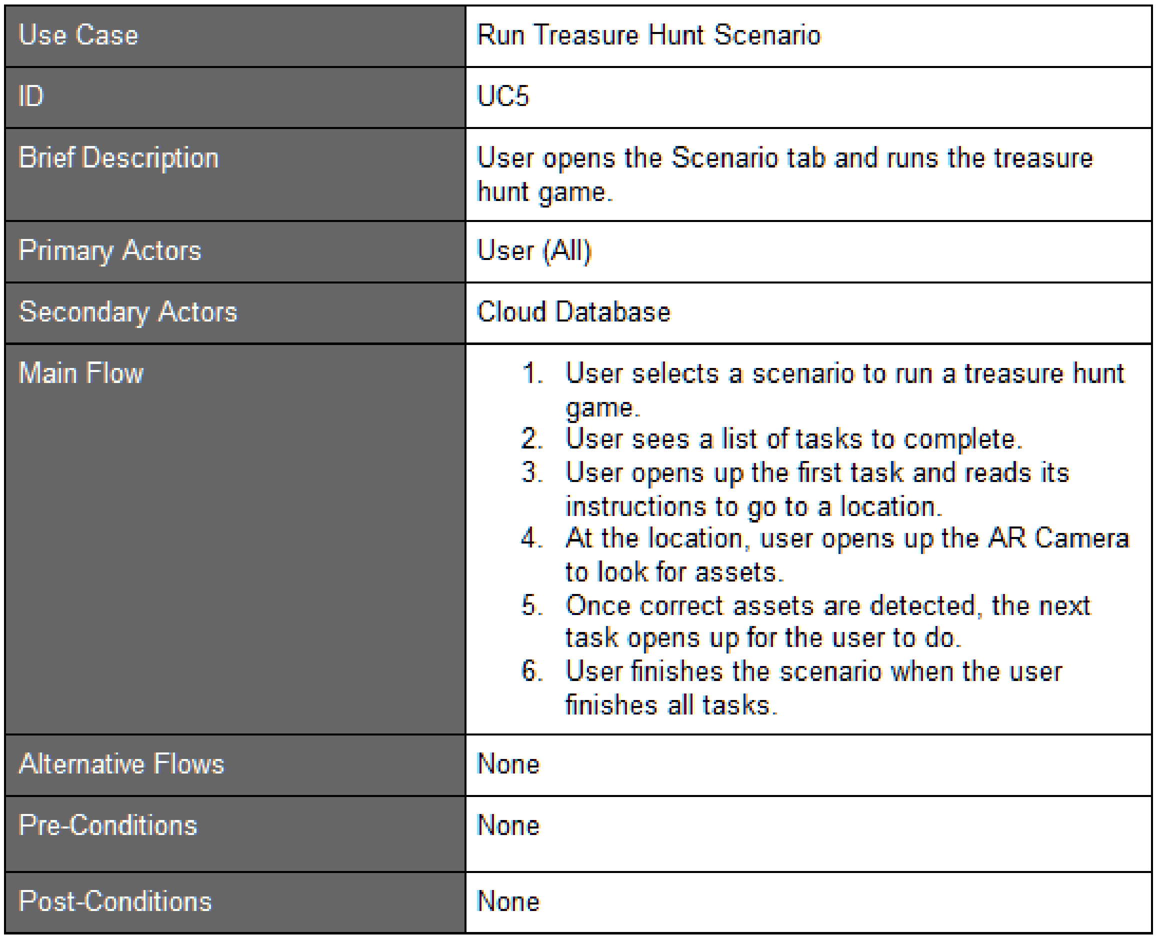
Use Case	Scan AR Asset
ID	UC3
Brief Description	App opens the AR Camera and begins detecting for assets located in front of the camera.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	1. User opens up the AR Camera 2. User points the AR Camera at surroundings. **If** assets exists at the location: 3. Phone displays assets through the AR Camera.
Alternative Flows	When running a scenario, the assets displayed are only the ones set by the scenario.
Pre-Conditions	None
Post-Conditions	Achievements page updated with list of things collected.

Table C.9: Detailed Use Case: Scan AR Asset



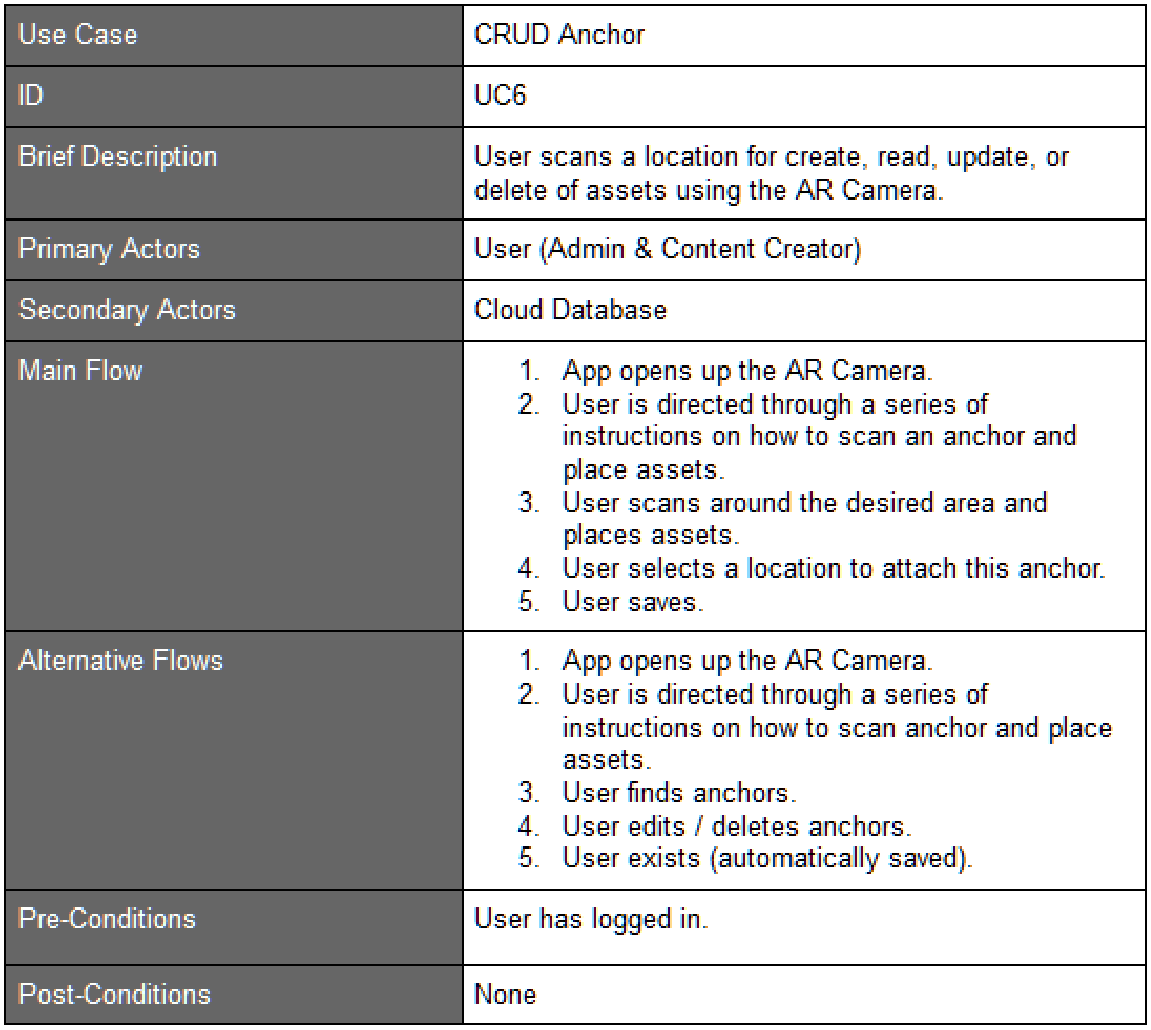
Use Case	View Achievements
ID	UC4
Brief Description	User selects the achievement tab and views collected assets of all locations.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	1. App displays list of locations and their scanned AR Assets. 2. User selects the AR Asset icons to view the assets in a larger picture or to run animations.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.10: Detailed Use Case: View Achievements



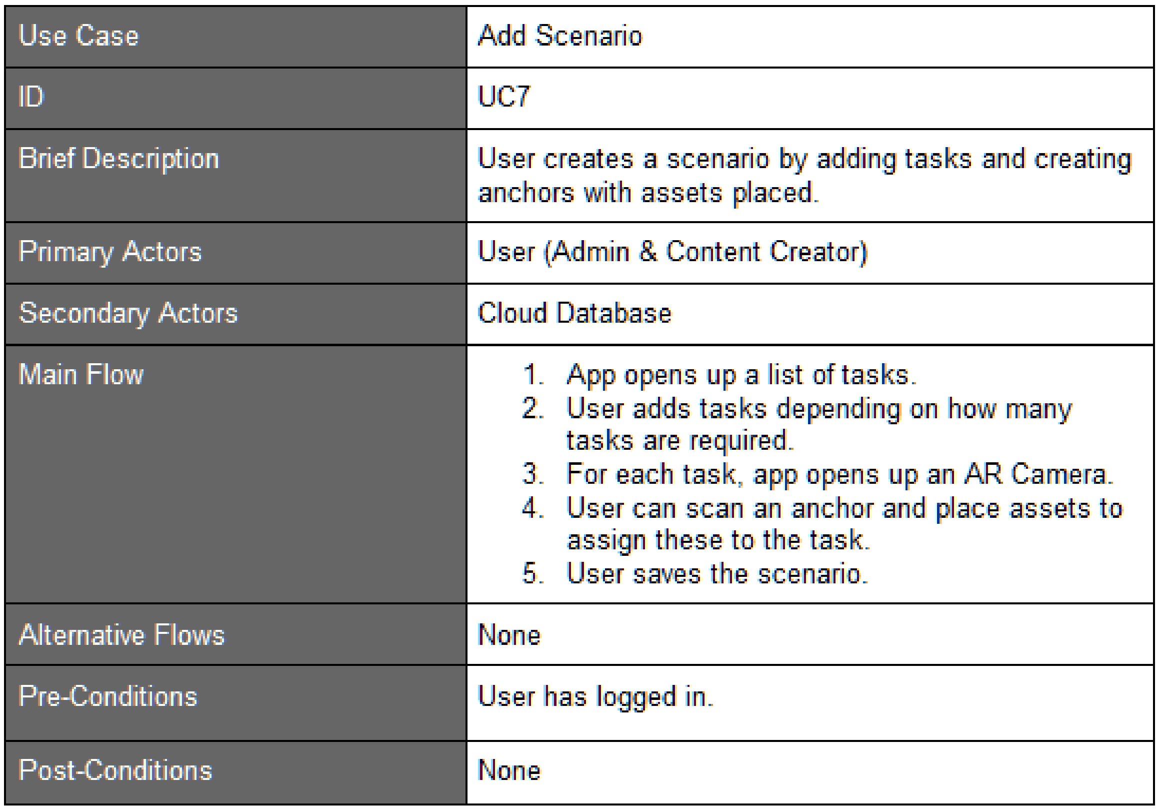
Use Case	Run Treasure Hunt Scenario
ID	UC5
Brief Description	User opens the Scenario tab and runs the treasure hunt game.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	1. User selects a scenario to run a treasure hunt game. 2. User sees a list of tasks to complete. 3. User opens up the first task and reads its instructions to go to a location. 4. At the location, user opens up the AR Camera to look for assets. 5. Once correct assets are detected, the next task opens up for the user to do. 6. User finishes the scenario when the user finishes all tasks.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.11: Detailed Use Case: Run Treasure Hunt Scenario



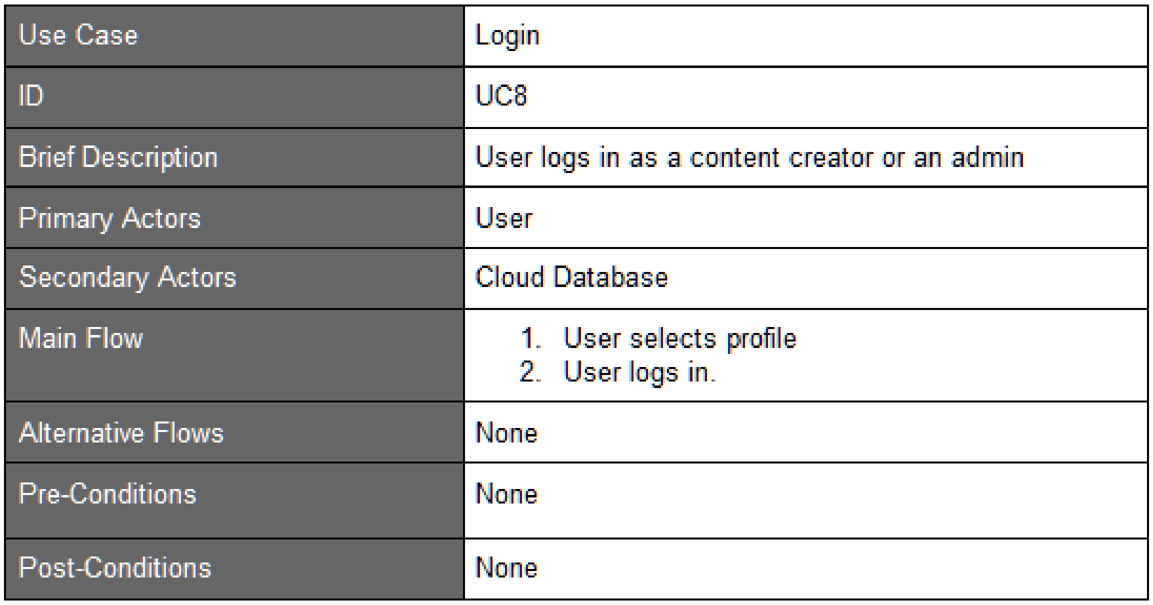
Use Case	CRUD Anchor
ID	UC6
Brief Description	User scans a location for create, read, update, or delete of assets using the AR Camera.
Primary Actors	User (Admin & Content Creator)
Secondary Actors	Cloud Database
Main Flow	1. App opens up the AR Camera. 2. User is directed through a series of instructions on how to scan an anchor and place assets. 3. User scans around the desired area and places assets. 4. User selects a location to attach this anchor. 5. User saves.
Alternative Flows	1. App opens up the AR Camera. 2. User is directed through a series of instructions on how to scan anchor and place assets. 3. User finds anchors. 4. User edits / deletes anchors. 5. User exists (automatically saved).
Pre-Conditions	User has logged in.
Post-Conditions	None

Table C.12: Detailed Use Case: CRUD Anchor



Use Case	Add Scenario
ID	UC7
Brief Description	User creates a scenario by adding tasks and creating anchors with assets placed.
Primary Actors	User (Admin & Content Creator)
Secondary Actors	Cloud Database
Main Flow	1. App opens up a list of tasks. 2. User adds tasks depending on how many tasks are required. 3. For each task, app opens up an AR Camera. 4. User can scan an anchor and place assets to assign these to the task. 5. User saves the scenario.
Alternative Flows	None
Pre-Conditions	User has logged in.
Post-Conditions	None

Table C.13: Detailed Use Case: Add Scenario



Use Case	Login
ID	UC8
Brief Description	User logs in as a content creator or an admin
Primary Actors	User
Secondary Actors	Cloud Database
Main Flow	1. User selects profile 2. User logs in.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

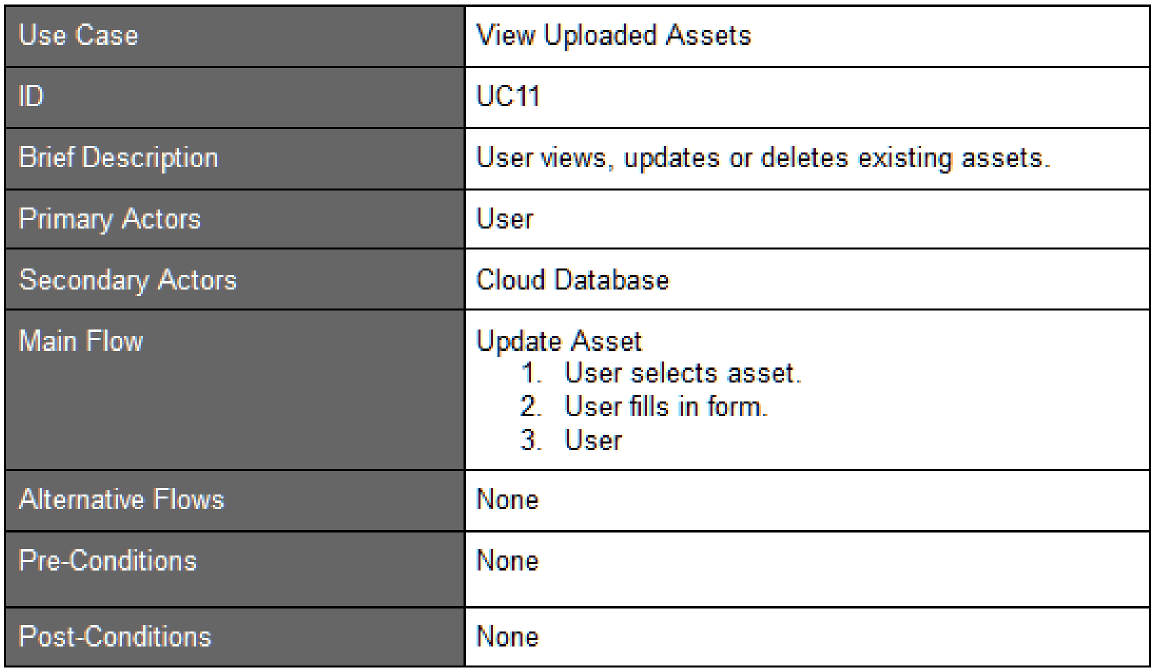
Table C.14: Detailed Use Case: App User Login

Use Case	Login
ID	UC9
Brief Description	User logs in as a content creator or an admin.
Primary Actors	User
Secondary Actors	Cloud Database
Main Flow	1. User enters website. 2. User fills in details. 3. User logs in.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.15: Detailed Use Case: Website User Login

Use Case	View Existing Anchors
ID	UC10
Brief Description	User view existing anchors and their locations.
Primary Actors	User (Admin)
Secondary Actors	Cloud Database
Main Flow	1. User selects appropriate tab. 2. User views a table of anchors and their locations.
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.16: Detailed Use Case: View Existing Anchors

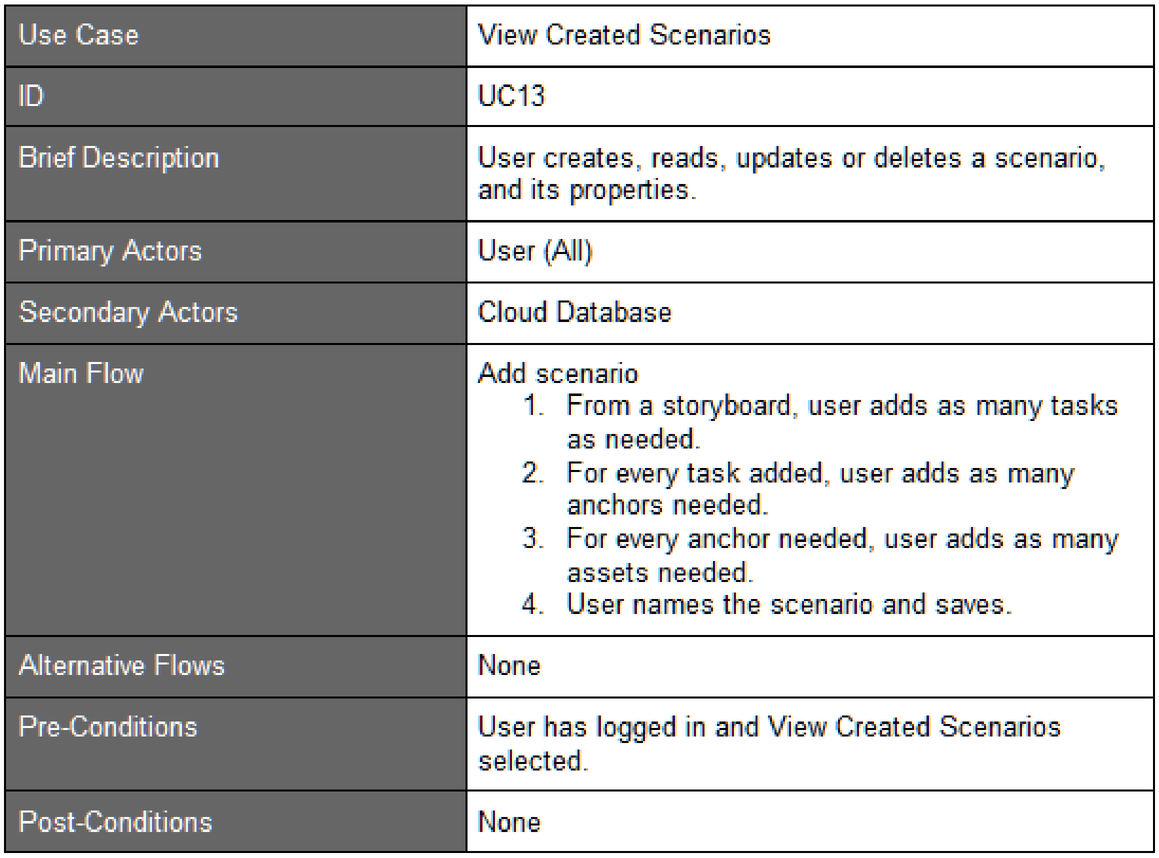


Use Case	View Uploaded Assets
ID	UC11
Brief Description	User views, updates or deletes existing assets.
Primary Actors	User
Secondary Actors	Cloud Database
Main Flow	Update Asset 1. User selects asset. 2. User fills in form. 3. User
Alternative Flows	None
Pre-Conditions	None
Post-Conditions	None

Table C.17: Detailed Use Case: View Uploaded Assets

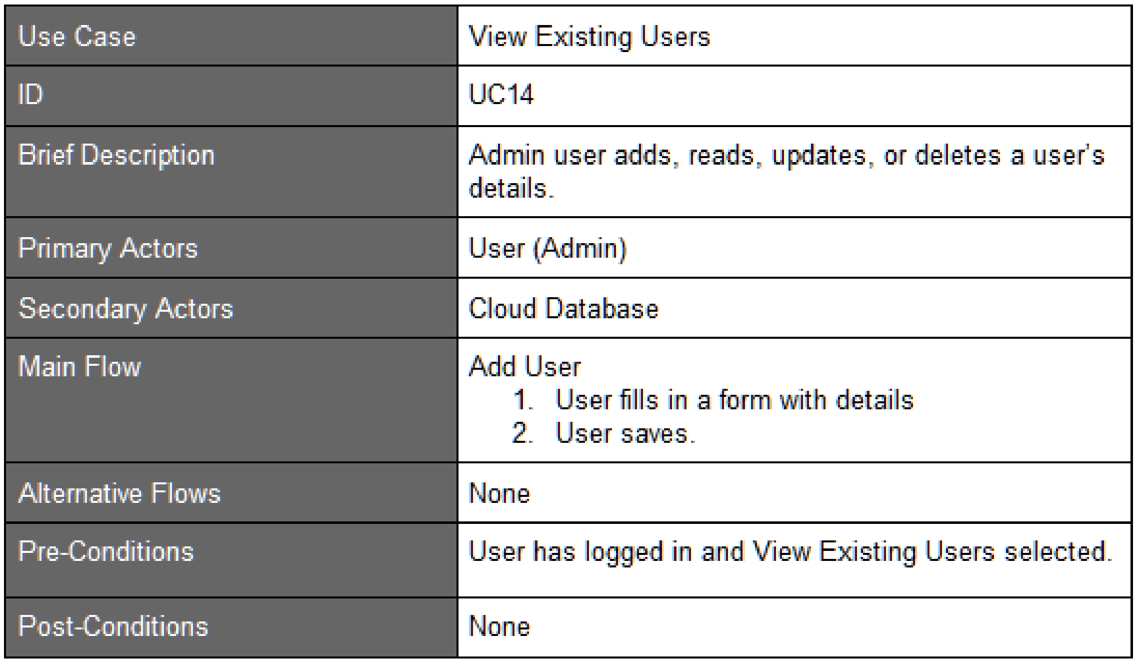
Use Case	View Uploaded Maps
ID	UC12
Brief Description	User views uploaded maps and can create, upload, read, or delete maps, locations and floors
Primary Actors	User (Admin)
Secondary Actors	Cloud Database
Main Flow	Add Map 1. User selects map name and file. 2. User sets properties of the map - if it is connected to a base map (map of buildings), if it is connected to a building, and if it is connected to a floor. 3. User creates Locations by highlighting areas on the map. 4. Highlighted areas are assigned names and location type of Building / Room. **If** location is of type Building: 5. User selects number of floors for this location. 6. User saves the map.
Alternative Flows	None
Pre-Conditions	User has logged in and View Uploaded Maps selected.
Post-Conditions	None

Table C.18: Detailed Use Case: View Uploaded Maps



Use Case	View Created Scenarios
ID	UC13
Brief Description	User creates, reads, updates or deletes a scenario, and its properties.
Primary Actors	User (All)
Secondary Actors	Cloud Database
Main Flow	Add scenario 1. From a storyboard, user adds as many tasks as needed. 2. For every task added, user adds as many anchors needed. 3. For every anchor needed, user adds as many assets needed. 4. User names the scenario and saves.
Alternative Flows	None
Pre-Conditions	User has logged in and View Created Scenarios selected.
Post-Conditions	None

Table C.19: Detailed Use Case: View Created Scenarios



Use Case	View Existing Users
ID	UC14
Brief Description	Admin user adds, reads, updates, or deletes a user's details.
Primary Actors	User (Admin)
Secondary Actors	Cloud Database
Main Flow	Add User 1. User fills in a form with details 2. User saves.
Alternative Flows	None
Pre-Conditions	User has logged in and View Existing Users selected.
Post-Conditions	None

Table C.20: Detailed Use Case: View Existing Users

Routes	**HTTP Request Method**	**Description**
/api/map-management/maps	GET	Gets all maps from the database, or perform a query with a map name or whether it is a default type.
/api/map-management/maps/:mapId	GET	Gets a map with mapId from the database.
/api/map-management/maps	POST	Add a map to the database and blob storage.
/api/map-management/maps/:id	PUT	Updates a map with the specified id on the database.
/api/map-management/maps/:id	DELETE	Deletes a map with the specified id on the database and blob storage.
/api/map-management/maps/:mapId/locations	GET	Gets all locations from the database, or query with a type filter.
/api/location-management/locations/:locationId	GET	Get a location with the specified id from the database.
/api/map-management/maps/:mapId/locations	POST	Add a location to the database.
/api/map-management/maps/:mapId/locations	DELETE	Delete a location from the database.
/api/map-management/maps/:mapId/locations/:buildingId/floors	GET	Get all floors from the database.
/api/floor-management/floors	GET	Get all floors from the database (used to perform a query with mapId).
/api/map-management/maps/:mapId/locations/:buildingId/floors	POST	Add a floor for a building.
/api/map-management/maps/:mapId/locations/:buildingId/floors/:floorId	UPDATE	Update a floor for a building.

/api/anchors	GET	Gets all anchors from the database.
/api/anchors/last	GET	Gets the last anchor added to the database.
/api/anchors/:anchorNumber	GET	Gets an anchor with anchorNumber from the database.
/api/anchors	POST	Uploads an anchor to the database.
/api/anchors	DELETE	Deletes an anchor from the database.
/api/asset-management/assets	GET	Gets all assets from the database (used with a query of type).
/api/asset-management/assets/:assetId	GET	Gets an asset with assetId from the database.
/api/asset-management/assets	POST	Uploads an asset with assetId to the database and blob storage.
/api/asset-management/assets/:assetId	PUT	Updates an asset with assetId from the database.
/api/asset-management/assets/:assetId	DELETE	Deletes an asset with assetId from the database and blob storage.
/api/asset-management/anchoredassets	GET	Get all anchored assets (used with a query for assetID) from the database.
/api/asset-management/anchoredassets	POST	Uploads anchored assets to the database.
/api/asset-management/anchoredassets	DELETE	Deletes anchored assets from the database (used with a query for anchoredassetID).

Table C.21: List of server routes with their respective HTTP requests and functional use description.

Appendix D

Code Listing

Below code listing is not the complete listing:

1. initialisation.SQL

```
DROP DATABASE IF EXISTS app_mysql;

CREATE DATABASE IF NOT EXISTS app_mysql;

USE app_mysql;

CREATE TABLE IF NOT EXISTS asset_types (
        asset_typeID INT AUTO_INCREMENT,
    asset_type_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (asset_typeID)
) ENGINE=INNODB;

INSERT INTO asset_types (asset_type_name) VALUES ('task');
INSERT INTO asset_types (asset_type_name) VALUES ('augmented_reality');
INSERT INTO asset_types (asset_type_name) VALUES ('map');

CREATE TABLE IF NOT EXISTS assets (
        assetID INT AUTO_INCREMENT,
    asset_typeID INT,
    asset_name VARCHAR(100) NOT NULL,
    blob_name VARCHAR(100),
    created_at DATETIME NOT NULL,
    is_default_map BOOL DEFAULT FALSE,
    PRIMARY KEY (assetID),
    FOREIGN KEY (asset_typeID) REFERENCES asset_types(asset_typeID) ON DELETE SET NULL ON UPDATE CASCADE
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS scenarios (
        scenarioID INT AUTO_INCREMENT,
        scenario_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (scenarioID)
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS tasks (
        taskID INT AUTO_INCREMENT,
        scenarioID INT NOT NULL,
    background_photoID INT,
    task_number INT NOT NULL,
    PRIMARY KEY (taskID),
    FOREIGN KEY (scenarioID) REFERENCES scenarios(scenarioID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (background_photoID) REFERENCES assets(assetID) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS task_display_photos (
        task_display_photoID INT AUTO_INCREMENT,
    taskID INT NOT NULL,
    display_photoID INT,
    PRIMARY KEY (task_display_photoID),
    FOREIGN KEY (taskID) REFERENCES tasks(taskID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (display_photoID) REFERENCES assets(assetID) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS location_types (
        location_typeID INT AUTO_INCREMENT,
    location_type_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (location_typeID)
) ENGINE=INNODB;

INSERT INTO location_types (location_type_name) VALUES ('building');
INSERT INTO location_types (location_type_name) VALUES ('room');
INSERT INTO location_types (location_type_name) VALUES ('location_connection');

CREATE TABLE IF NOT EXISTS locations (
        locationID INT AUTO_INCREMENT,
    current_mapID INT NOT NULL,
    location_typeID INT NOT NULL,
    location_name VARCHAR (100) NOT NULL,
    rel_position_on_map_x DECIMAL(10,5),
    rel_position_on_map_y DECIMAL(10,5),
    PRIMARY KEY (locationID),
    FOREIGN KEY (current_mapID) REFERENCES assets(assetID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (location_typeID) REFERENCES location_types(location_typeID) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS anchors (
        anchorNumber INT AUTO_INCREMENT,
        anchorID VARCHAR(100),
    locationID INT,
    anchor_name VARCHAR(100),
    PRIMARY KEY (anchorNumber),
    FOREIGN KEY (locationID) REFERENCES locations(locationID) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE IF NOT EXISTS anchored_assets (
        anchored_assetID INT AUTO_INCREMENT,
    anchorNumber INT NOT NULL,
    assetID INT NOT NULL,
    PRIMARY KEY (anchored_assetID),
    FOREIGN KEY (anchorNumber) REFERENCES anchors(anchorNumber) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS anchored_assets_tasks (
        anchored_assets_taskID INT AUTO_INCREMENT,
    taskID INT NOT NULL,
    anchored_assetID INT NOT NULL,
    PRIMARY KEY (anchored_assets_taskID),
    FOREIGN KEY (taskID) REFERENCES tasks(taskID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (anchored_assetID) REFERENCES anchored_assets(anchored_assetID) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS location_floor_maps (
        location_floor_mapID INT AUTO_INCREMENT,
    selected_locationID INT NOT NULL,
    floor_mapID INT,
    floor_number INT NOT NULL,
    PRIMARY KEY (location_floor_mapID),
    FOREIGN KEY (selected_locationID) REFERENCES locations(locationID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (floor_mapID) REFERENCES assets(assetID) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS location_connection_maps (
        location_connection_mapID INT AUTO_INCREMENT,
    selected_locationID INT NOT NULL,
    connection_mapID INT,
    PRIMARY KEY (location_connection_mapID),
    FOREIGN KEY (selected_locationID) REFERENCES locations(locationID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (connection_mapID) REFERENCES assets(assetID) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS user_types (
        user_typeID INT AUTO_INCREMENT,
    user_type_name VARCHAR(100),
    PRIMARY KEY (user_typeID)
) ENGINE=INNODB;

INSERT INTO user_types (user_type_name) VALUES ('admin');
INSERT INTO user_types (user_type_name) VALUES ('content_creator');
INSERT INTO user_types (user_type_name) VALUES ('public');

CREATE TABLE IF NOT EXISTS users (
        userID INT AUTO_INCREMENT,
    user_typeID INT NOT NULL,
    user_username VARCHAR(100) NOT NULL,
    user_password VARCHAR(100) NOT NULL,
    PRIMARY KEY (userID),
    FOREIGN KEY (user_typeID) REFERENCES user_types(user_typeID) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS user_collected_assets (
        user_collected_assetID INT AUTO_INCREMENT,
    userID INT NOT NULL,
    collected_assetID INT NOT NULL,
    collected_at_anchorNumber INT NOT NULL,
    PRIMARY KEY (user_collected_assetID),
    FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (collected_assetID) REFERENCES assets(assetID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (collected_at_anchorNumber) REFERENCES anchors(anchorNumber) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB;
```

2. server.js

```
// REQUIRE STATEMENTS

const express = require('express');
const bodyParser = require('body-parser');
const multer = require('multer');
const inMemoryStorage = multer.memoryStorage();
const uploadStrategy = multer({ storage: inMemoryStorage }).single('file_path'); // when uploading files, have to s
const uploadFile = multer({ storage: inMemoryStorage }).single('file');
const anchors = require('./anchors');
const assets = require('./assets');
const anchoredAssets = require('./anchoredassets');
const maps = require('./maps');
const locations = require('./locations');
const floors = require('./floors');

// APP START

const app = express();

const whitelist = ['http://localhost:4200', 'https://app-treasure-hunt-website.azurewebsites.net'];

app.use(function(req, res, next) {
    let selectedOrigin = null;
    let origin = req.get('origin');
    console.log('origin: ', origin)
    if (whitelist.indexOf(origin) >= 0 || !origin) { // if origin is in whitelisted origins
        selectedOrigin = origin
    }
    res.header("Access-Control-Allow-Origin", selectedOrigin); // update to match the domain you will make the requ
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
    res.header("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT, DELETE");
    next();
});

app.use(bodyParser.json()); // support JSON encoded bodies
app.use(bodyParser.urlencoded({ extended: true }));

app.listen((process.env.PORT || 8000), () => {
    console.log('Server started on port 8000')
})

// --- MAPS --- //
// GET ALL MAPS AS A TABLE OR QUERY WITH NAME / DEFAULT
app.get('/api/map-management/maps', maps.getAllMaps);

// GET A MAP WITH AN ID AND ITS BLOB
app.get('/api/map-management/maps/:mapId', maps.getMap);

// ADD A MAP
app.post('/api/map-management/maps', uploadStrategy, maps.addMap);

// UPDATE A MAP
app.put('/api/map-management/maps/:id', uploadStrategy, maps.updateMap);

// DELETE MAP
app.delete('/api/map-management/maps/:id', uploadStrategy, maps.deleteMap);

// --- LOCATIONS --- //

// GET ALL LOCATIONS WITH OPTIONAL TYPE OF A MAP OR QUERY IT
app.get('/api/map-management/maps/:mapId/locations', locations.getAllLocationsByMapId);

// GET ALL LOCATIONS
app.get('/api/location-management/locations', locations.getAllLocations)

// GET A LOCATION WITH AN ID
app.get('/api/location-management/locations/:locationId', locations.getLocation);

// Add a location
app.post('/api/map-management/maps/:mapId/locations', locations.addLocation);

// DELETE ALL LOCATIONS
app.delete('/api/map-management/maps/:mapId/locations', uploadStrategy, locations.deleteAllLocations);

// --- FLOORS --- //

// GET ALL FLOORS WITH OPTIONAL TYPE OF A BUILDING
app.get('/api/map-management/maps/:mapId/locations/:buildingId/floors', floors.getAllFloors);

// GET FLOOR FROM MAPID AS QUERY
app.get('/api/floor-management/floors', floors.getFloor);

// add a floor for a building
app.post('/api/map-management/maps/:mapId/locations/:buildingId/floors', floors.addFloor);

// update a floor for a building
app.put('/api/map-management/maps/:mapId/locations/:buildingId/floors/:floorId', floors.updateFloor);


// --- ANCHORS --- //
app.get('/api/anchors', anchors.getAllAnchorKeysAsync);
app.get('/api/anchors/last', anchors.getLastAnchorKeyAsync);
app.get('/api/anchors/:anchorNumber', anchors.getAnchorKeyAsync);
app.post('/api/anchors', anchors.setAnchorKeyAsync);
app.delete('/api/anchors', anchors.deleteAnchorKeyAsync);

// ---- ASSETS --- //
app.get('/api/asset-management/assets', assets.getAllAssetsOfType);
app.get('/api/asset-management/assets/:assetId', assets.getAssetWithId)
app.post('/api/asset-management/assets', uploadFile, assets.setNonMapAsset);

app.delete('/api/asset-management/assets/:assetId', uploadStrategy, assets.deleteAsset);


// ---- ANCHORED ASSETS ---//
app.get('/api/asset-management/anchoredassets', anchoredAssets.getAnchoredAssetsAsync);
app.post('/api/asset-management/anchoredassets', anchoredAssets.setAnchoredAssetAsync);
app.delete('/api/asset-management/anchoredassets', anchoredAssets.deleteAnchoredAssetAsync);

// For testing
module.exports = app;

// Error detection
app.use(function (err, req, res, next) {
    console.log('This is the invalid field ->' + err.field);
```

```
        next(err)
    })
```

3. database.js

```
const mysql = require('mysql');

module.exports = class Database {
    constructor(config) {
        if(!!Database.instance){
            console.log('returning singleton');
            return Database.instance;
        }

        console.log('creating a singleton');
        Database.instance = this;

        try {
            this.pool = mysql.createPool(config);
        } catch(err) {
            console.log("Error connecting to database pool: " + err);
        }
    }

    query(sql, args) {
        return new Promise( (resolve, reject) => {
            this.pool.query(sql, args, (err, rows) => {
                if (err) {
                    return reject(err);
                } else {
                    resolve(rows);
                };
            });
        });
    };
}
```

4. maps.js

```
const getStream = require('into-stream');
const blob_access = require('./blob_access');

const config = require('./config');
const environment = config.environment;
const Database = require('./database.js');
const database = new Database(config[environment].database);
const STORAGE_ACCOUNT_NAME = config[environment].storage.accountName;

const getBlobName = originalName => {
    // Use a random number to generate a unique file name,
    // removing "0." from the start of the string.
    const identifier = Math.random().toString().replace(/0\./, '');
    return `${identifier}-${originalName}`;
};

module.exports.getAllMaps = function(req, res) {
    // connect_to_database(connection);
    console.log('Getting maps...');

    const containerName = "assets";

    if (Object.keys(req.query).length === 0) {
        let sql = `SELECT assetID, asset_type_name, asset_name, blob_name, created_at, is_default_map
                    FROM assets AS a1
                    LEFT JOIN asset_types AS a2
                    ON a1.asset_typeID = a2.asset_typeID
                    WHERE a1.asset_typeID = a2.asset_typeID
                    AND a2.asset_type_name = "map";`

        database.query(sql).then(rows => {
            for (const row of rows) {
                const blobName = row.blob_name;
                const getBlobUrl = 'https://' + STORAGE_ACCOUNT_NAME + '.blob.core.windows.net' + '/' + containerNar
                row.imgUrl = getBlobUrl;
            }
            console.log('sending rows');
            res.send(rows);
        }, err => {
            console.log("Error in getting maps, error: " + err);
            throw new Error();
        // }).then( () => {
        //     database.close_connection();
        }).catch( err => {
            console.log("Something went wrong ... " + err);
            res.status(400).send('Error in database operation - Get all maps');
        });

    } else {
        let name = req.query.name;
        if (name) {
            let sql = `SELECT * FROM assets AS a1, asset_types AS a2
                    WHERE a1.asset_typeID = a2.asset_typeID
                    AND a2.asset_type_name = "map"
                    AND a1.asset_name = ?;`

            database.query(sql, [name]).then(rows => {
                console.log('Retrieved map name successfully');
                res.send(rows);
            }, err => {
                console.log("Error in getting maps with query, error: " + err);
                throw new Error();
            }).catch( err => {
                console.log("Something went wrong ... ");
                res.status(400).send('Error in database operation - Get all maps with query');
            });
        } else {
            let is_default_map = req.query.default;
            let sql = `SELECT assetID, asset_type_name, asset_name, blob_name, created_at, is_default_map FROM asse
                        LEFT JOIN asset_types AS a2
                        ON a1.asset_typeID = a2.asset_typeID
                        WHERE a1.asset_typeID = a2.asset_typeID
                        AND a2.asset_type_name = "map"
                        AND a1.is_default_map = ?;`

            console.log('IS DEFAULT MAP: ' + is_default_map);

            if(is_default_map) {
                is_default_map = 1;
            } else {
                is_default_map = 0;
            }
            database.query(sql, [is_default_map]).then(rows => {
                for (const row of rows) {
                    const blobName = row.blob_name;
                    const getBlobUrl = 'https://' + STORAGE_ACCOUNT_NAME + '.blob.core.windows.net' + '/' + containe
                    row.imgUrl = getBlobUrl;
                }
                console.log('Retrieved default successfully');
                res.send(rows);
            }, err => {
                console.log("Error in getting default map with query, error: " + err);
                throw new Error();
            }).catch( err => {
                console.log("Something went wrong ... ");
                res.status(400).send('Error in database operation - Get maps with query');
            });
        }
    }
}

module.exports.getMap = function(req,res) {
    let mapId = req.params.mapId;
    let containerName = "assets";

    let sql = `SELECT * FROM assets AS a1, asset_types AS a2
                WHERE a1.asset_typeID = a2.asset_typeID
                AND a2.asset_type_name = "map"
                AND a1.assetID = ?;`

    database.query(sql, [mapId]).then(rows => {
        if (rows[0]) {
            const blobName = rows[0].blob_name;
            const getBlobUrl = 'https://' + STORAGE_ACCOUNT_NAME + '.blob.core.windows.net' + '/' + containerName +
            rows[0].imgUrl = getBlobUrl;
        }
        res.send(rows);
    }, err => {
        console.log("Error in getting a map with ID, error: " + err);
        throw new Error();
    // }).then( () => {
    //     database.close_connection();
    }).catch( err => {
        console.log("Something went wrong ... " + err);
        res.status(400).send('Error in database operation - Get map with ID');
    });
}

module.exports.getAllLocations = function(req, res) {
    let mapId = req.params.mapId;
    // test out if req.query is empty what happens, basically use a query string to indicate type

    // if no query parameters
    if (Object.keys(req.query).length === 0) {
        let sql = `SELECT l1.locationID, l1.current_mapID, l1.location_typeID, l1.location_name, l1.rel_position_on_
                   FROM locations AS l1
                   JOIN location_types AS lt
                   ON l1.location_typeID = lt.location_typeID
                   AND l1.current_mapID = ?
                   LEFT JOIN location_floor_maps as lfm
                   ON l1.locationID = lfm.selected_locationID
                   ORDER BY l1.locationID, lfm.floor_number;`

        database.query(sql, [mapId]).then(rows => {
            res.send(rows);
        }, err => {
            console.log("Error in getting all locations of a map, error: " + err);
            throw new Error();
        }).catch( err => {
            console.log("Something went wrong ... ");
            res.status(400).send('Error in database operation - Get all locations of a map');
        });

    } else {
        let name = req.query.name;
        let type = req.query.type;

        if (name) {
            console.log('getting location with name...');
            let sql = `SELECT * FROM locations AS l1, location_types AS l2
                       WHERE l1.location_typeID = l2.location_typeID
                       AND l2.location_type_name = ?
                       AND l1.current_mapID = ?
                       AND l1.location_name = ?;`
            database.query(sql, [type, mapId, name]).then(rows => {
                res.send(rows);
            }, err => {
                console.log('Error in getting location name of a map, error: ' + err);
                throw new Error();
            }).catch(err => {
                console.log("Something went wrong ... ");
                res.status(400).send('Error in database operation - Get location of a map with name');
            })

        } else {
            let sql = `SELECT * FROM locations AS l1, location_types AS l2
                       WHERE l1.location_typeID = l2.location_typeID
                       AND l2.location_type_name = ?
                       AND l1.current_mapID = ?;`

            database.query(sql, [type, mapId]).then(rows => {
                res.send(rows);
            }, err => {
                console.log("Error in getting location types of a map, error: " + err);
                throw new Error();
            }).catch( err => {
                console.log("Something went wrong ... ");
                res.status(400).send('Error in database operation - Get location types of a map');
            });
        }
    }
}

module.exports.addMap = async function (req, res) {
    /* Obtain form values */

    let map_name = req.body.file_name;
    let is_default_map = req.body.is_default_map;
    console.log('IS DEFAULT MAP = ' + is_default_map);
    if(is_default_map == "true") {
        is_default_map = 1;
    } else {
        is_default_map = 0;
    }

    let stream = getStream(req.file.buffer);
    let blob_name = getBlobName(map_name);
    let location_floor_mapId = req.body.location_floor_mapID; // to connect the map id to the floor

    /* Upload map to blob storage */
    let container_name = "assets";

    try {
        /* upload a file to blob storage */
        await blob_access.uploadStream(container_name, stream, blob_name);
        console.log("Added map to blob storage successfully.")

    } catch {
        console.log('Error - Uploading asset to Blob Storage: ');
        console.log('---request body: ' + JSON.stringify(req.body));
        console.log('---container_name: ' + container_name);
        console.log('---map_name: ' + map_name);
        res.status(400).send('Error in uploading asset to Blob Storage: ');
    }

    /* Upload map to database */

    let sql_getID = `SELECT asset_typeID FROM asset_types WHERE asset_type_name = "map";`
    let selected_asset_typeID;

    let sql_addMap = `INSERT INTO assets (asset_typeID, asset_name, blob_name, created_at, is_default_map)
                        VALUES (?, ?, ?, CURRENT_TIMESTAMP, ?);`
    let sql_updateFloorWithMap = `UPDATE location_floor_maps
                                    SET floor_mapID = ?
                                    WHERE location_floor_mapID = ?;`

    database.query(sql_getID).then(rows => {
        /* First retrieve asset_typeID to be used to for adding to database later */
        /* Once retrieved, add map to database with the asset_typeID */
        selected_asset_typeID = rows[0].asset_typeID;
        console.log('Success - Retrieved asset_typeID for adding map to database: ' + selected_asset_typeID);
        return database.query(sql_addMap, [selected_asset_typeID, map_name, blob_name, is_default_map]);
    }, err => {
        console.log('Error in database operation - Retrieving asset_typeID for adding map to database.');
        throw new Error();
    }).then(rows => {
        console.log("Added map to database successfully.");
        let attached_mapId = rows.insertId;

        if (location_floor_mapId) {
            database.query(sql_updateFloorWithMap, [attached_mapId, location_floor_mapId]).then(rows => {
                console.log('Success - updated floor with map');
                res.status(200).send({text: 'Added map to database and floor successfully'});
            }, err => {
                console.log('Failed - add map to floor: ' + err);
                throw new Error();
            });
        } else {
            res.status(200).send({text: 'Added map to database successfully'});
        }
    }, err => {
        /* If error occured in adding to database, delete blob that has been uploaded to keep things consistent */
        console.log('Error in adding map to database');
        console.log('map_name: ' + map_name);
        console.log('blob_name: ' + blob_name);
        console.log('selected_asset_typeID: ' + selected_asset_typeID);
        console.log('if no errors then it is a sql problem');
        blob_access.deleteBlob(container_name, blob_name);
        throw new Error();
    }).catch( err => {
        res.status(400).send('Error in database operation - Add map.');
    })
}

module.exports.updateMap = async function(req, res) {
    let mapId = req.params.id;
    let container_name = "assets";

    console.log('updating a map req body: ' + JSON.stringify(req.body));
    console.log('updating a map req file body: ' + JSON.stringify(req.file));

    let name = req.body.name;
    let map_name = req.body.file_name;
    let blob_name = req.body.blob_name;
    let location_floor_mapId = req.body.location_floor_mapID
    let prev_location_floor_mapId = req.body.prev_location_floor_mapID

    // if changed selected floor
    if (prev_location_floor_mapId !== location_floor_mapId) {
        // set previous floor mapID to be null
        let sql_prev_floor = `UPDATE location_floor_maps SET floor_mapID = null
                              WHERE location_floor_mapID = ?;`

        // set new floor map ID to be this map's ID
        let sql_new_floor = `UPDATE location_floor_maps SET floor_mapID = ?
                              WHERE location_floor_mapID = ?;`

        database.query(sql_prev_floor, [prev_location_floor_mapId]).then(res => {
            return database.query(sql_new_floor, [mapId, location_floor_mapId])
        }, err => {
            console.log('Error - cannot set previous floor map id to be null: ' + err);
            throw new Error();
        }).then(res => {
            console.log('Success - Updated new floor map ID');
        }, err => {
            console.log('Error - cannot update new floor map Id');
            throw new Error();
        }).catch( err => {
            res.status(400).send('Error in database operation - update map.');
            return;
        });
    };

    // if uploaded file
    if (req.file) {
        let stream = getStream(req.file.buffer);
        let new_blob_name = getBlobName(map_name);
        blob_access.deleteBlob(container_name, blob_name); // delete old blob

        try {
            /* upload a file to blob storage */
            await blob_access.uploadStream(container_name, stream, blob_name);
            console.log("Added map to blob storage successfully.")

        } catch {
            console.log('Error - Uploading asset to Blob Storage: ');
            console.log('---request body: ' + JSON.stringify(req.body));
            console.log('---container_name: ' + container_name);
            console.log('---map_name: ' + map_name);
            res.status(400).send('Error in uploading asset to Blob Storage: ');
            return;
        }

        /* Upload map to database */

        let sql_updateMap = `UPDATE assets SET asset_name = ?, SET blob_name = ?
                              WHERE assetID = ?;`


        database.query(sql_updateMap, [name, new_blob_name, mapId]).then(rows => {
            // first update map details
            console.log('Success - Updated map for database: ');
            res.status(200).send({text: 'Added map to database successfully'});
        }, err => {
            console.log('Error in updating map: ' + err);
        }).catch( err => {
            res.status(400).send('Error in database operation - Add map.');
            return;
        })

    } else {
        let sql = 'UPDATE assets SET asset_name = ? WHERE assetID = ?;'
        database.query(sql, [name, mapId]).then(rows => {
            res.json({ message: 'Map updated!' });
        }).catch( err => {
            res.status(400).send('Error in database operation - update map.');
            return;
        });
    }
}

module.exports.deleteMap = async function(req, res) {
    let id = req.params.id;

    let sql_deleteMap = `DELETE FROM assets WHERE assetID = ?`;
    let sql_getMap = `SELECT blob_name FROM assets WHERE assetID = ?`;

    let container_name = "assets";

    let blob_name;

    database.query(sql_getMap, [id]).then(rows => {
        console.log('Got map');
        blob_name = rows[0].blob_name;
        return database.query(sql_deleteMap, [id]);
    }, err => {
        console.log("Failed in retrieving map" + err);
    }).then(rows => {
        console.log('Deleted map');
        return blob_access.deleteBlob(container_name, blob_name);
    }, err => {
        console.log('Cant delete map from database');
        throw new Error();
    }).then(rows => {
        res.status(200).send({
            event: 'successful deletion of map'
        })
        console.log("Deleted map from blob storage successfully.")
    }, err => {
        console.log('Error - Deleting asset from Blob Storage: ');
        throw new Error();
    }).catch( err => {
        res.status(400).send('Error in database operation - Add locations and floors.');
    });
}
```

5. maps-add-form.component.ts

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { ConfigService } from '../services/config.service';
import { FormGroup, FormControl } from '@angular/forms';
import { MatDialog } from '@angular/material/dialog';
import { LocationFormComponent } from '../location-form/location-form.component';
import { Location } from '../classes/location';
import { Router } from '@angular/router';
import { AddResponse } from '../classes/addResponse';


@Component({
  selector: 'app-maps-add-form',
  templateUrl: './maps-add-form.component.html',
  styleUrls: ['./maps-add-form.component.css']
})
export class MapsAddFormComponent implements OnInit, OnDestroy {
  subscription;
  buildingsSubscription;
  floorsSubscription;

  mapId: string;
  buildingId: string;

  maps;
  buildings;
  floors;

  addMapForm = new FormGroup({
    uploadedMapName: new FormControl(''),
    uploadedMapFile: new FormControl(''),
    uploadedMapDefault: new FormControl(''),
    selectedMap: new FormControl(''),
    selectedBuilding: new FormControl(''),
    selectedFloor: new FormControl('')
  });

  addMapSelectSubscription;
  buildingSelectSubscription;
  onSelectedMap = false;

  selectedFile: File;

  selectedIsMap = false;
  selectedIsBuilding = false;

  imgURL;

  locations: Location[];

  constructor(
    private configService: ConfigService,
    private dialog: MatDialog,
    private router: Router
  ) { }

  ngOnInit() {
    document.getElementById('addLocations').style.display = 'none';

    this.subscription = this.configService.getMaps().subscribe((res: Response) => {
      this.maps = JSON.parse(JSON.stringify(res));
    }, error => console.log(error));

    this.addMapSelectSubscription = this.addMapForm.get('selectedMap').valueChanges.subscribe(res => {
      this.mapId = this.addMapForm.get('selectedMap').value;
      console.log('found map ID ' + this.mapId);

      if (this.isMap(this.mapId)) {
        this.selectedIsMap = true;

        this.buildingsSubscription = this.configService.getLocationsOfMapWithType(this.mapId, 'building').subscribe
          this.buildings = JSON.parse(JSON.stringify(res2));
        }, error => console.log(error));

      } else {
        this.selectedIsMap = false;
      }
    });
    this.buildingSelectSubscription = this.addMapForm.get('selectedBuilding').valueChanges.subscribe(res => {
      this.buildingId = this.addMapForm.get('selectedBuilding').value;

      if (this.isBuilding(this.mapId, this.buildingId)) {
        this.selectedIsBuilding = true;

        this.floorsSubscription = this.configService.getFloorsWithoutMaps(this.mapId, this.buildingId).subscribe((r
          this.floors = JSON.parse(JSON.stringify(res2));
        }, error => console.log(error));

      } else {
        this.selectedIsBuilding = false;
      }
    });
  }

  async isMap(mapID) {
    // send mapID's get request
    const isMapSubscription = await this.configService.getMapWithID(mapID).subscribe((res) => {
      // isMapSubscription.unsubscribe();
      const resMap = JSON.parse(JSON.stringify(res))[0]; // response is a list with one element

      if (resMap) {
        if (resMap.asset_type_name === 'map') {
          return true;
        }
      }
      return false;
    });
  }

  async isBuilding(mapId, buildingID) {
    // send buildingID's get request
    const isBuildingSubscription = await this.configService.getLocationWithID(buildingID).subscribe((res: Response)
      isBuildingSubscription.unsubscribe();
      const location = JSON.parse(JSON.stringify(res))[0]; // response is a list with one element

      if (location.location_type_name === 'building') {
        return true;
      } else {
        return false;
      }
    });
  }

  onSubmitAddMap() {
    console.log(this.addMapForm.value);
    document.getElementById('addMapForm').style.display = 'none';
    document.getElementById('addLocations').style.display = 'block';

    const fileReader = new FileReader();
    fileReader.onload = (event => {
      this.imgURL = fileReader.result;
      this.initCanvas();
    });
    fileReader.readAsDataURL(this.selectedFile);
  }

  onFileSelected(event) {
    const file = event.target.files[0];
    this.selectedFile = file;
  }

  displayForm() {
    document.getElementById('addMapForm').style.display = 'block';
    document.getElementById('addLocations').style.display = 'none';
  }

  drawCanvas(ctx, widthCanvas, heightCanvas, widthMouse, heightMouse) {
    // clear canvas
    ctx.clearRect(0, 0, widthCanvas, heightCanvas);

    // draw image in background
    const img = new Image();
  console.log('image is ' + this.imgURL);
  img.src = this.imgURL;
  img.onload = (res) => {
    ctx.drawImage(img, 0, 0, widthCanvas, heightCanvas);

    // draw existing locations
    console.log('this.locations: ' + JSON.stringify(this.locations));
    for (const location of this.locations) {
      const x = location.relativePositionOnMap.x * widthCanvas;
      const y = location.relativePositionOnMap.y * heightCanvas;

      ctx.fillStyle = 'red';
      ctx.fillRect(x, y, widthMouse, heightMouse);
    }
  };
}

initCanvas() {
  const canvas = document.getElementById('mapCanvas') as HTMLCanvasElement;
  const ctx = canvas.getContext('2d');
  const rect = canvas.getBoundingClientRect();
  const widthMouse = 15;
  const heightMouse = 15;

  const widthCanvas = 1000;
  const heightCanvas = 700;
  canvas.setAttribute('width', String(widthCanvas));
  canvas.setAttribute('height', String(heightCanvas));

  this.locations = [];
  let location;

  this.drawCanvas(ctx, widthCanvas, heightCanvas, widthMouse, heightMouse);

  // draw a Location when user clicks
  canvas.addEventListener('mousedown', (event) => {
    const x = event.clientX - rect.left - (widthMouse / 2);
    const y = event.clientY - rect.top - (heightMouse / 2);

    // check if a Location has been drawn at the spot
    location = this.isWithinHitRegionLocation(this.locations, widthMouse, heightMouse, x, y, widthCanvas, heightC
    let dialog;
    if (location) {
      console.log('opening an existing location');
      console.log('location: ' + location);
      dialog = this.openLocationDialog(location);
      dialog.afterClosed().subscribe(data => {
        if (data.event === 'delete') {
          console.log('Delete Location');
          const deleteIndex = this.locations.indexOf(location, 0);
          if (deleteIndex > -1) {
            this.locations.splice(deleteIndex, 1);
            this.drawCanvas(ctx, widthCanvas, heightCanvas, widthMouse, heightMouse);
          }
        }
      });

    } else {
      ctx.fillStyle = 'red';
      ctx.fillRect(x, y, widthMouse, heightMouse);

      console.log('x: ' + x);
      console.log('y: ' + y);

      const relPositionOnMapX = x / widthCanvas;
      const relPositionOnMapY = y / heightCanvas;

      location = new Location(relPositionOnMapX, relPositionOnMapY);
      // since location is new, add a location to this.locations
      dialog = this.openLocationDialog(location);
      dialog.afterClosed().subscribe(data => {
        if (data) {
          if (data.event === 'delete') {
            console.log('Delete Location');
            const deleteIndex = this.locations.indexOf(location, 0);
            if (deleteIndex > -1) {
              this.locations.splice(deleteIndex, 1);
              this.drawCanvas(ctx, widthCanvas, heightCanvas, widthMouse, heightMouse);
            }
          }
        }

        this.locations.push(location);
        console.log('added locations: ', this.locations);
      });
    }


  });
}

openLocationDialog(location: Location) {
  return this.dialog.open(LocationFormComponent, {
    width: '500px',
    height: '500px',
    data: location
  });
}

isWithinHitRegionLocation(
  locations: Location[],
  locationWidth: number,
  locationHeight: number,
  mouseX: number,
  mouseY: number,
  canvasWidth: number,
  canvasHeight: number) {

    for (const location of locations) {
      const locationX = location.relativePositionOnMap.x * canvasWidth;
      const locationY = location.relativePositionOnMap.y * canvasHeight;
      const locationLeft = locationX - locationWidth;
      const locationRight = locationX + locationWidth;
      const locationTop = locationY - locationHeight;
      const locationBottom = locationY + locationHeight;

      if (mouseX >= locationLeft) {
        if (mouseX <= locationRight) {
          // mouse is in between region horizontally
          if (mouseY >= locationTop) {
            if (mouseY <= locationBottom) {
              // mouse is in between region vertically
              return location;
            }
          }
        }
      }
    }
    return false;
  // get existing locations
}

async onSubmitForm() {
  console.log('Submitting Form');

  const formData = new FormData();
  const mapName = this.addMapForm.get('uploadedMapName').value;
  const mapDefault = this.addMapForm.get('uploadedMapDefault').value;
  const locationFloorMapID = this.addMapForm.get('selectedFloor').value;
  console.log("Default map: " + mapDefault);
  formData.append('file_name', mapName);
  formData.append('is_default_map', mapDefault);
  formData.append('file_path', this.selectedFile);
  formData.append('location_floor_mapID', locationFloorMapID);

  console.log('adding map form data');
  await this.configService.addMap(formData).subscribe(res => {
    console.log(res);
    let lastMapId: string;
    console.log('getting recently added map id');

    this.configService.getMapWithName(mapName).subscribe(res2 => {

      console.log(mapName);
      console.log(res);
      lastMapId = res2[0].assetID;

      console.log('adding location form data');

      if (this.locations.length === 0) {
        this.router.navigate(['/maps']);
      }

      for (const location of this.locations) {

        console.log('location is: ' + JSON.stringify(location));

        const loc = {
          name: location.name,
          type: location.type.type,
          relativePositionOnMapX: location.relativePositionOnMap.x,
          relativePositionOnMapY: location.relativePositionOnMap.y,
        };

        const floors = [];
        if (Location.determineIfIsBuildingOrRoom(location.type)) {
          for (const lFloor of location.type.floors) {
            const floor = {
              type: 'floor',
              floorNumber: ''
            };
            floor.floorNumber = String(lFloor.floorNumber);
            floors.push(floor);
          }
        }

        console.log('floors: ' + JSON.stringify(floors));
        console.log('lastMapId: ' + lastMapId);
        console.log('loc: ' + JSON.stringify(loc));

        this.configService.addLocation(String(lastMapId), loc).subscribe((res3: AddResponse) => {
          console.log('Added location');
          console.log('response: ' + JSON.stringify(res3));

          const lastBuildingId = res3.lastInsertedId;
          console.log('last building id: ' + lastBuildingId);

          for (const floor of floors) {
            console.log('adding floors...');
            this.configService.addFloor(String(lastMapId), lastBuildingId, floor).subscribe(res5 => {
              console.log('added floor ' + floor.floorNumber);
            });
          }
          this.router.navigate(['/maps']);
        }, err => {
          console.log('Cant add locations: ' + err);
        });
      }
    });
  });
}



ngOnDestroy() {
}

// ngOnDestroy() {
//   this.addMapSelectSubscription.unsubscribe();
//   this.buildingSelectSubscription.unsubscribe();
```

```
    //    this.subscription.unsubscribe();
    //    this.buildingsSubscription.unsubscribe();
    //    this.floorsSubscription.unsubscribe();
    // }
}
```

6. AdminCrudAnchorsController.cs

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT license.
using System.Linq;
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.Threading.Tasks;

namespace Microsoft.Azure.SpatialAnchors.Unity.Examples
{
    public class AdminCrudAnchorsController : ArAppScriptBase
    {
        internal enum AppState
        {
            StepChooseFlow = 0,
            StepInputAnchorNumber,
            StepCreateSession,
            StepConfigSession,
            StepStartSession,
            StepCreateLocalAnchor,
            StepSaveCloudAnchor,
            StepSavingCloudAnchor,
            StepStopSession,
            StepDestroySession,
            StepCreateSessionForQuery,
            StepStartSessionForQuery,
            StepLookForAnchor,
            StepLookingForAnchor,
            StepLookedForAnchor,
            StepDeletedAnchor,
            StepStopSessionForQuery,
            StepComplete,
        }

        internal enum Flow
        {
            CreateFlow = 0,
            LocateFlow
        }

        private readonly Dictionary<AppState, StepParams> stateParams = new Dictionary<AppState, StepParams>
        {
            { AppState.StepChooseFlow,new StepParams() { StepMessage = "Next: Start your flow", StepColor = Color.c
            { AppState.StepInputAnchorNumber,new StepParams() { StepMessage = "Next: Input anchor number", StepColo
            { AppState.StepCreateSession,new StepParams() { StepMessage = "Next: Create CloudSpatialAnchorSession",
            { AppState.StepConfigSession,new StepParams() { StepMessage = "Next: Configure CloudSpatialAnchorSessio
            { AppState.StepStartSession,new StepParams() { StepMessage = "Next: Start CloudSpatialAnchorSession", S
            { AppState.StepCreateLocalAnchor,new StepParams() { StepMessage = "Tap a surface to add the local ancho
            { AppState.StepSaveCloudAnchor,new StepParams() { StepMessage = "Next: Save local anchor to cloud", Ste
            { AppState.StepDeletedAnchor, new StepParams() { StepMessage = "Next: Create local anchor", StepColor =
            { AppState.StepSavingCloudAnchor,new StepParams() { StepMessage = "Saving local anchor to cloud...", St
            { AppState.StepStopSession,new StepParams() { StepMessage = "Next: Stop cloud anchor session", StepColo
            { AppState.StepDestroySession,new StepParams() { StepMessage = "Next: Destroy Cloud Anchor session", St
            { AppState.StepCreateSessionForQuery,new StepParams() { StepMessage = "Next: Create CloudSpatialAnchorS
            { AppState.StepStartSessionForQuery,new StepParams() { StepMessage = "Next: Start CloudSpatialAnchorSes
            { AppState.StepLookForAnchor,new StepParams() { StepMessage = "Next: Look for anchor", StepColor = Colo
            { AppState.StepLookingForAnchor,new StepParams() { StepMessage = "Looking for anchor..press next to sto
            { AppState.StepLookedForAnchor,new StepParams() { StepMessage = "Looked for anchor...", StepColor = Col
            { AppState.StepStopSessionForQuery,new StepParams() { StepMessage = "Next: Stop CloudSpatialAnchorSessi
            { AppState.StepComplete,new StepParams() { StepMessage = "Next: Restart ", StepColor = Color.clear }}
        };

        //#if !UNITY_EDITOR
        public AnchorExchanger anchorExchanger = new AnchorExchanger();
        //#endif

        #region Member Variables
        private AppState _currentAppState = AppState.StepChooseFlow;
        private Flow _currentFlow = Flow.CreateFlow;
        private readonly List<GameObject> otherSpawnedObjects = new List<GameObject>();
        private int anchorsLocated = 0;
        private int anchorsExpected = 0;
        private readonly List<string> localAnchorIds = new List<string>();
        private string _anchorKeyToFind = null;
        private long? _anchorNumberToFind;
        private AnchorList anchorList;
        #endregion // Member Variables

        #region Unity Inspector Variables
        [SerializeField]
        [Tooltip("The base URL for the example sharing service.")]
        private string baseSharingUrl = "";
        #endregion // Unity Inspector Variables

        private AppState currentAppState
        {
            get
            {
                return _currentAppState;
            }
            set
            {
                if (_currentAppState != value)
                {
                    Debug.LogFormat("State from {0} to {1}", _currentAppState, value);
                    _currentAppState = value;
                    if (spawnedObjectMat != null)
                    {
                        spawnedObjectMat.color = stateParams[_currentAppState].StepColor;
                    }

                    feedbackBox.text = stateParams[_currentAppState].StepMessage;
                    EnableCorrectUIControls();
                }
            }
        }

        protected override void OnCloudAnchorLocated(AnchorLocatedEventArgs args)
        {
            base.OnCloudAnchorLocated(args);

            if (args.Status == LocateAnchorStatus.Located)
            {
                //CloudSpatialAnchor nextCsa = args.Anchor;
                //currentCloudAnchor = args.Anchor;

                UnityDispatcher.InvokeOnAppThread(async () =>
                {
                    CloudSpatialAnchor nextCsa = args.Anchor;
                    currentCloudAnchor = args.Anchor;

                    anchorsLocated++;
                    Debug.Log("Anchor located! Total anchors located: " + anchorsLocated);
                    currentCloudAnchor = nextCsa;
                    Pose anchorPose = Pose.identity;

                    Debug.Log("Found anchor: " + nextCsa.Identifier);

                    #if UNITY_ANDROID || UNITY_IOS
                    anchorPose = nextCsa.GetPose();
#endif
                    // HoloLens: The position will be set based on the unityARUserAnchor that was located.

                    // Get anchor from anchorID (cloudIdentifier)
                    Debug.Log("Retrieve anchor key with id");
                    AnchorList retrievedAnchorList = await anchorExchanger.RetrieveAnchorKeyWithId(nextCsa.Identifi
                    Anchor retrievedAnchor = null;
                    AnchoredAsset retrievedAnchoredAsset = null;
            Asset retrievedAsset= null;
            Debug.Log("Check if retrieved anchorlist exists");
            if (retrievedAnchorList.anchors.Length > 0)
            {
                Debug.Log("Retrievedanchorlist anchors length is: " + retrievedAnchorList.anchors.Length);
                retrievedAnchor = retrievedAnchorList.anchors[0];


                // Get anchored asset
                Debug.Log("Getting anchored asset: with anchorNumber" + retrievedAnchor.anchorNumber);
                Debug.Log("And Anchor key is: " + retrievedAnchor.anchorID);
                AnchoredAssetList retrievedAnchoredAssetList = await service.GetAnchoredAssetWithAnchorNumb

                Debug.Log("Check if there is anchored asset");
                if (retrievedAnchoredAssetList.anchoredAssets.Length > 0)
                {
                    Debug.Log("Retrieved anchoredasset");
                    retrievedAnchoredAsset = retrievedAnchoredAssetList.anchoredAssets[0];

                    var assetService = AddAssetManager.GetComponent<AssetService>();
                    Debug.Log("Get asset prefab");
                    AssetList retrievedAssetList = await service.GetARAssetWithAssetID(retrievedAnchoredAss
                    if (retrievedAssetList.assets.Length > 0)
                    {
                        Debug.Log("Retrieved asset");
                        retrievedAsset = retrievedAssetList.assets[0];
                    }
                }
            }
            Debug.Log("Spawning object with id: " + nextCsa.Identifier);
            Debug.Log("Spawning object with anchor number: " + retrievedAnchor.anchorNumber);
            GameObject nextObject = SpawnNewAnchoredObject(anchorPose.position,
                                                           anchorPose.rotation,
                                                           nextCsa,
                                                           retrievedAnchor,
                                                           retrievedAnchoredAsset,
                                                           retrievedAsset);
            if (nextObject.transform.GetChild(0).GetChild(0).TryGetComponent<MeshRenderer>(out var tempMesh
            {
                spawnedObjectMat = tempMesh.material;
            }

            otherSpawnedObjects.Add(nextObject);

            //if (anchorsLocated >= anchorsExpected)
            //{
            //    currentAppState = AppState.StepLookedForAnchor;
            //}
            // currentAppState = AppState.StepLookedForAnchor;
        });
    }
}

/// <summary>
/// Start is called on the frame when a script is enabled just before any
/// of the Update methods are called the first time.
/// </summary>
public override void Start()
{
    base.Start();

    //var button = XRUXPickerForSharedAnchorDemo.Instance.GetSaveButton();
    //button.onClick.RemoveAllListeners();
    //button.onClick.AddListener(() =>
    //{
    //    currentAppState = AppState.StepSaveCloudAnchor;
    //});

    if (!SanityCheckAccessConfiguration())
    {
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[1].gameObject.SetActive(false);
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(false);
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoInputField().gameObject.SetActive(false);
        return;
    }

    SpatialAnchorSamplesConfig samplesConfig = Resources.Load<SpatialAnchorSamplesConfig>("SpatialAnchorSam
    if (string.IsNullOrWhiteSpace(BaseSharingUrl) && samplesConfig != null)
    {
        BaseSharingUrl = samplesConfig.BaseSharingURL;
    }

    if (string.IsNullOrEmpty(BaseSharingUrl))
    {
        feedbackBox.text = $"Need to set {nameof(BaseSharingUrl)}.";
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[1].gameObject.SetActive(false);
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(false);
        XRUXPickerForSharedAnchorDemo.Instance.GetDemoInputField().gameObject.SetActive(false);
        return;
    }
    else
    {
        Uri result;
        if (!Uri.TryCreate(BaseSharingUrl, UriKind.Absolute, out result))
        {
            feedbackBox.text = $"{nameof(BaseSharingUrl)} is not a valid url";
            return;
        }
        else
        {
            BaseSharingUrl = $"{result.Scheme}://{result.Host}/api/anchors";
        }
    }

    //#if !UNITY_EDITOR
    anchorExchanger.WatchKeys(BaseSharingUrl);
    //#endif

    feedbackBox.text = stateParams[currentAppState].StepMessage;

    Debug.Log("Azure Spatial Anchors Shared  script started");
    EnableCorrectUIControls();
}

/// <summary>
/// Update is called every frame, if the MonoBehaviour is enabled.
/// </summary>
protected override void Update()
{
    base.Update();

    if (spawnedObjectMat != null)
    {
        float rat = 0.1f;
        float createProgress = 0f;
        if (CloudManager.SessionStatus != null)
        {
            createProgress = CloudManager.SessionStatus.RecommendedForCreateProgress;
        }
        rat += (Mathf.Min(createProgress, 1) * 0.9f);
        spawnedObjectMat.color = stateParams[currentAppState].StepColor * rat;
    }
}

protected override bool IsCreatingAnchorObject()
{
    return currentAppState == AppState.StepCreateLocalAnchor;
}

protected override Color GetStepColor()
{
    if (currentCloudAnchor == null || localAnchorIds.Contains(currentCloudAnchor.Identifier))
    {
        return stateParams[currentAppState].StepColor;
    }

    return Color.magenta;
}
```

```
#pragma warning disable CS1998 // Conditional compile statements are removing await
        protected override async Task OnSaveCloudAnchorSuccessfulAsync()
#pragma warning restore CS1998

        {
            Debug.Log("On save cloud anchor successful async");
            await base.OnSaveCloudAnchorSuccessfulAsync();

            long anchorNumber = -1;

            localAnchorIds.Add(currentCloudAnchor.Identifier);

            // save to anchor database
            Debug.Log("Saving to anchor database");
            var anchor = new Anchor();
            anchor.anchorID = currentCloudAnchor.Identifier;
            anchorNumber = await anchorExchanger.StoreAnchorKey(anchor);

            // update anchormodel information
            Debug.Log("Adding anchor model information: and anchor number is: " + anchorNumber);
            spawnedObject.GetComponent<AnchorModel>().anchor.anchorID = anchor.anchorID;
            spawnedObject.GetComponent<AnchorModel>().anchor.anchorNumber = (int)anchorNumber;
            spawnedObject.GetComponent<AnchorModel>().anchoredAsset.anchorNumber = (int)anchorNumber;

            // save to anchoredassets database
            Debug.Log("Saving to anchored asset database");
            var anchoredAsset = spawnedObject.GetComponent<AnchorModel>().anchoredAsset;

            if (anchoredAsset.assetID > 0)
            {
                Debug.Log("Uploading anchored asset: " + anchoredAsset.assetID);
                var response = await service.UploadAnchoredAsset(anchoredAsset);
                if (response != null)
                {
                    Debug.Log("Uploaded anchored asset");
                }
                else
                {
                    Debug.Log("Couldn't upload anchored asset");
                }
            }

            Pose anchorPose = Pose.identity;

            #if UNITY_ANDROID || UNITY_IOS
            anchorPose = currentCloudAnchor.GetPose();
#endif
            // MoveObject(spawnedObject, anchorPose.position, anchorPose.rotation); // move spawnedObject

            //AttachTextMesh(spawnedObject, anchorNumber);
            //Debug.Log("spawnedobject.transform.getchild(0).getchild(0): " + spawnedObject.transform.GetChild(0).G

            if (spawnedObject.transform.GetChild(0).GetChild(0).gameObject.TryGetComponent<MeshRenderer>(out var ab
            {
                Debug.Log("spawnedobject.transform.getchild(0).getchild(0) is an asset ");
            } else
            {
                MeshRenderer bcd = spawnedObject.transform.GetChild(0).GetChild(0).gameObject.GetComponentInChildre
                if (bcd != null)
                {
                    Debug.Log("spawnedobject.transform.getchild(0).getchild(0).getcomponentinchildren is an asset")
                }

            }

            // feedbackBox.text = $"Created anchor {anchorNumber}. Next: Stop cloud anchor session";
            feedbackBox.text = $"Created anchor {anchorNumber}. ";
        }

        protected override async Task OnDeleteCloudAnchorSuccessfulAsync()
        {
            Debug.Log("Delete from azure successful!");
            await base.OnDeleteCloudAnchorSuccessfulAsync();
        // Delete from anchor database
        await anchorExchanger.DeleteAnchorKey(spawnedObject.GetComponent<AnchorModel>().anchor);

        // Delete from anchoredassets database, no need to do this yet because we are only attaching one anchor
        // await service.DeleteAnchoredAsset(spawnedObject.GetComponentInChildren<AnchoredAsset>());

        feedbackBox.text = $"Deleted anchor.";

    }

    protected override void OnSaveCloudAnchorFailed(Exception exception)
    {
        base.OnSaveCloudAnchorFailed(exception);
    }



    public async override Task AdvanceAsync()
    {
        if (currentAppState == AppState.StepChooseFlow || currentAppState == AppState.StepInputAnchorNumber)
        {
            return;
        }

        if (_currentFlow == Flow.CreateFlow)
        {
            await AdvanceCreateFlowAsync();
        }
        else if (_currentFlow == Flow.LocateFlow)
        {
            await AdvanceLocateFlowAsync();
        }
    }

    public async Task InitializeCreateFlowAsync()
    {
        Debug.Log("Initialize Create Flow  Async");
        Debug.Log("Appstate step choose flow: " + AppState.StepChooseFlow);
        if (currentAppState == AppState.StepChooseFlow)
        {
            Debug.Log("Setting createflow");
            _currentFlow = Flow.CreateFlow;

            // get all anchor numbers
            anchorList = await anchorExchanger.RetrieveAllAnchorKeys();
            currentAppState = AppState.StepCreateSession;
        }
        else
        {
            Debug.Log("Advance  async");
            await AdvanceAsync();
        }
    }

    // To be used for delete button click
    public async void DeleteAnchor()
    {
        Debug.Log("Trying to Delete Anchor");
        try
        {
            Debug.Log("Checking if spawnedObject is null");
            if (spawnedObject != null)
            {
                Debug.Log("Deleting anchor...");
                var currentAnchor = spawnedObject.GetComponent<AnchorModel>().anchor;
                if (currentAnchor.anchorID != null)
                {
                    await DeleteCurrentObjectAnchorToCloudAsync();
                    CleanupSpawnedObject();
                }
                else
                {
                    CleanupSpawnedObject();
                    }
                }
            }
            catch (Exception ex)
            {
                Debug.LogError("Failed to delete current object anchor: " + ex);
            }

        }

        /// <summary>
        /// This version only exists for Unity to wire up a button click to.
        /// If calling from code, please use the Async version above.
        /// </summary>
        public async void InitializeCreateFlow()
        {
            try
            {
                await InitializeCreateFlowAsync();
            }
            catch (Exception ex)
            {
                Debug.LogError($"{nameof(AdminCrudAnchorsController)} - Error in {nameof(InitializeCreateFlow)}: {e
            }
        }


#pragma warning disable CS1998 // Conditional compile statements are removing await

        public async Task InitializeLocateFlowAsync()
#pragma warning restore CS1998
        {
            if (currentAppState == AppState.StepChooseFlow)
            {
                currentAppState = AppState.StepInputAnchorNumber;
            }
            else if (currentAppState == AppState.StepInputAnchorNumber)
            {
                //long anchorNumber;

                // get all anchor numbers
                anchorList = await anchorExchanger.RetrieveAllAnchorKeys();

                if (anchorList.anchors.Length == 0)
                {
                    feedbackBox.text = "No Anchor Numbers!";

                }
                else
                {
                    //_anchorNumberToFind = anchorNumber;

                    //#if !UNITY_EDITOR
                    //AnchorList anchorList = await anchorExchanger.RetrieveAnchorKey(_anchorNumberToFind.Value);
                    //#endif

                    _currentFlow = Flow.LocateFlow;
                    currentAppState = AppState.StepCreateSession;
                    XRUXPickerForSharedAnchorDemo.Instance.GetDemoInputField().text = "";

                    //if (anchorList == null)
                    //{
                    //     anchor = null;
                    //     _anchorKeyToFind = null;
                    //     feedbackBox.text = "Anchor Number Not Found!";
                    //}
                    //else
                    //{
                    //     anchor = anchorList.anchors[0];
                    //     _anchorKeyToFind = anchor.anchorID.ToString();
                    //     _currentFlow = Flow.LocateFlow;
                    //     currentAppState = AppState.StepCreateSession;
                    //     XRUXPickerForSharedAnchorDemo.Instance.GetInputField().text = "";
                    //}
            }
        }
        else
        {
            await AdvanceAsync();
        }
    }

    /// <summary>
    /// This version only exists for Unity to wire up a button click to.
    /// If calling from code, please use the Async version above.
    /// </summary>
    public async void InitializeLocateFlow()
    {
        try
        {
            await InitializeLocateFlowAsync();
        }
        catch (Exception ex)
        {
            Debug.LogError($"{nameof(AdminCrudAnchorsController)} - Error in {nameof(InitializeLocateFlow)}: {e
        }
    }

    public override async void ReturnToLauncher()
    {
        CloudManager.StopSession();
        CleanupSpawnedObjects();
        await CloudManager.ResetSessionAsync();
        base.ReturnToLauncher();
    }


    // Button Click to save current selected object
    public async void SaveCurrentObjectAnchorToCloud()
    {
        Debug.Log("Pressing save button");
        if (currentAppState == AppState.StepCreateLocalAnchor)
        {
            Debug.Log("Saving cloud anchor...");
            currentAppState = AppState.StepSaveCloudAnchor;
            await SaveCurrentObjectAnchorToCloudAsync();
            Debug.Log("Change back to create local anchor state");
            currentAppState = AppState.StepCreateLocalAnchor;
        }
    }

    private async Task AdvanceCreateFlowAsync()
    {
        switch (currentAppState)
        {
            case AppState.StepCreateSession:
                currentCloudAnchor = null;
                currentAppState = AppState.StepCreateSessionForQuery;
                break;
            case AppState.StepCreateSessionForQuery:
                anchorsLocated = 0;
                ConfigureSession();
                Debug.Log("Changing to start session query..");
                currentAppState = AppState.StepStartSessionForQuery;
                break;
            case AppState.StepStartSessionForQuery:
                Debug.Log("Start session async");
                await CloudManager.StartSessionAsync();
                if (anchorList.anchors.Length == 0)
                {
                    currentAppState = AppState.StepCreateLocalAnchor;
                } else
                {
                    currentAppState = AppState.StepLookForAnchor;
                }
                break;
            case AppState.StepLookForAnchor:
                currentAppState = AppState.StepLookingForAnchor;
                currentWatcher = CreateWatcher();
                break;
            case AppState.StepLookingForAnchor:
                currentWatcher.Stop();
                // when user presses next, stop looking for anchors

                currentAppState = AppState.StepCreateLocalAnchor;
                break;
            case AppState.StepLookedForAnchor:
                currentAppState = AppState.StepCreateLocalAnchor;
                break;
            case AppState.StepCreateLocalAnchor: // user creates and edits anchors
                break;
            case AppState.StepDeletedAnchor:
                currentAppState = AppState.StepCreateLocalAnchor;
                break;
            case AppState.StepSaveCloudAnchor: // must be used by 'save' button
                currentAppState = AppState.StepSavingCloudAnchor;
                //foreach (var toSaveObj in spawnedObjects)
                //{
                //     spawnedObject = toSaveObj;
                //     await SaveCurrentObjectAnchorToCloudAsync();
                //}
                //await SaveCurrentObjectAnchorToCloudAsync();
                break;
            case AppState.StepStopSession:
                CloudManager.StopSession();
                CleanupSpawnedObjects();
                await CloudManager.ResetSessionAsync();
                // currentAppState = AppState.StepComplete;
                break;
            case AppState.StepComplete:
                currentCloudAnchor = null;
                currentAppState = AppState.StepChooseFlow;
                CleanupSpawnedObjects();
                break;
            default:
                Debug.Log("Shouldn't get here for app state " + currentAppState);
                break;
        }
    }

    private async Task AdvanceLocateFlowAsync()
    {
        switch (currentAppState)
        {
            case AppState.StepCreateSession:
                currentAppState = AppState.StepChooseFlow;
                currentCloudAnchor = null;
                currentAppState = AppState.StepCreateSessionForQuery;
                break;
            case AppState.StepCreateSessionForQuery:
                anchorsLocated = 0;
                ConfigureSession();
                currentAppState = AppState.StepStartSessionForQuery;
                break;
            case AppState.StepStartSessionForQuery:
                await CloudManager.StartSessionAsync();
                currentAppState = AppState.StepLookForAnchor;
                break;
            case AppState.StepLookForAnchor:
                currentAppState = AppState.StepLookingForAnchor;
                currentWatcher = CreateWatcher();
                break;
            case AppState.StepLookingForAnchor:

                // Advancement will take place when anchors have all been located.
                break;
            case AppState.StepLookedForAnchor:
                currentAppState = AppState.StepStopSessionForQuery;
                break;
            case AppState.StepStopSessionForQuery:
                CloudManager.StopSession();
                currentAppState = AppState.StepComplete;
                break;
            case AppState.StepComplete:
                currentCloudAnchor = null;
                currentWatcher = null;
                currentAppState = AppState.StepChooseFlow;
                CleanupSpawnedObjects();
                break;
            default:
                Debug.Log("Shouldn't get here for app state " + currentAppState);
                break;
        }
    }

    private void EnableCorrectUIControls()
    {
        switch (currentAppState)
        {
            case AppState.StepChooseFlow:

                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[1].gameObject.SetActive(false);
                #if UNITY_WSA
                XRUXPickerForSharedAnchorDemo.Instance.transform.position = Camera.main.transform.position + Ca
                XRUXPickerForSharedAnchorDemo.Instance.transform.LookAt(Camera.main.transform);
                XRUXPickerForSharedAnchorDemo.Instance.transform.Rotate(Vector3.up, 180);
                XRUXPickerForSharedAnchorDemo.Instance.GetButtons()[0].gameObject.SetActive(true);
                #else
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].transform.Find("Text").GetComponent<
                #endif
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoInputField().gameObject.SetActive(false);
                break;
            case AppState.StepInputAnchorNumber:
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[1].gameObject.SetActive(true);
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(false);
                //XRUXPickerForSharedAnchorDemo.Instance.GetInputField().gameObject.SetActive(true);
                break;
            case AppState.StepCreateLocalAnchor:
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(false);
                break;
            default:
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[1].gameObject.SetActive(false);
#if UNITY_WSA
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(false);
#else
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].gameObject.SetActive(true);
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoButtons()[0].transform.Find("Text").GetComponent<
                #endif
                XRUXPickerForSharedAnchorDemo.Instance.GetDemoInputField().gameObject.SetActive(false);
                break;
        }
    }

    private void ConfigureSession()
    {
        Debug.Log("Configure session...");
        List<string> anchorsToFind = new List<string>();

        if (currentAppState == AppState.StepCreateSessionForQuery)
        {
            Debug.Log("For each anchor...");
            foreach (var anchor in anchorList.anchors)
            {
                Debug.Log("Each anchor id: " + anchor.anchorID);
                _anchorKeyToFind = anchor.anchorID.ToString();
                anchorsToFind.Add(_anchorKeyToFind);
            }
        }

        anchorsExpected = anchorsToFind.Count;
        SetAnchorIdsToLocate(anchorsToFind);
    }

    protected override void CleanupSpawnedObjects()
    {
        base.CleanupSpawnedObjects();
```

```
            for (int index = 0; index < otherSpawnedObjects.Count; index++)
            {
                Destroy(otherSpawnedObjects[index]);
            }

            otherSpawnedObjects.Clear();
        }

        /// <summary>
        /// Gets or sets the base URL for the example sharing service.
        /// </summary>
        public string BaseSharingUrl { get => baseSharingUrl; set => baseSharingUrl = value; }
    }
}
```

7. CreateAssetBundles.cs

```
  using UnityEditor;
using System.IO;
using UnityEngine;

public class CreateAssetBundles
{
    static string assetBundleDirectory = "Assets/AssetBundles";

    [MenuItem("Assets/Build and Upload AssetBundles")]
    static void BuildAndUploadAllAssetBundles()
    {
        BuildAllAssetBundles();
        UploadAllAssetBundles();
    }

    [MenuItem("Assets/Build AssetBundles")]
    static void BuildAllAssetBundles()
    {

        if (!Directory.Exists(assetBundleDirectory))
        {
            Directory.CreateDirectory(assetBundleDirectory);
        }
        //BuildPipeline.BuildAssetBundles(assetBundleDirectory, BuildAssetBundleOptions.None, BuildTarget.Standalone

        BuildPipeline.BuildAssetBundles(assetBundleDirectory, BuildAssetBundleOptions.None, BuildTarget.Android);
    }

    static async void UploadAllAssetBundles()
    {

        DatabaseService service = DatabaseService.Instance;

        string[] filePaths = Directory.GetFiles(@".\Assets\AssetBundles");
        foreach (var file in filePaths)
        {
            if (!file.Contains("Assets\\AssetBundles\\AssetBundles"))
            {
                if (!(file.EndsWith(".meta") || file.EndsWith(".manifest")))
                {
                    //Debug.Log("File Path uploading... " + file);
                    //service.Upload(file, Path.GetFileName(file));
                    //Debug.Log("Done")
                    string response = await service.uploadAssetBundle(file, Path.GetFileName(file));
                    Debug.Log("Response: " + response);
                }
                Debug.Log("Deleting Filename is: " + file);
                File.Delete(file);
            }
        }
    }
}
```